

Владимир Попов

САМОУЧИТЕЛЬ

Паскаль и Дельфи

```
ParamCount <> 2 then  
begin  
  WriteLn('Bad command line')  
  Halt(1);  
end  
else  
begin  
  ReadFile(ParamStr(1))  
  WriteFile(ParamStr(2))  
end
```

Прочитав
эту книгу,
вы научитесь:

- ◆ общим принципам программирования
- ◆ основным элементам языка Паскаль
- ◆ алгоритмам и методам программирования
- ◆ основам объектно-ориентированного программирования
- ◆ порядку разработки собственных приложений



 ПИТЕР

СЕРИЯ

САМОУЧИТЕЛЬ

32.34
152
Владимир Попов

САМОУЧИТЕЛЬ

Паскаль и Дельфи

 **ПИТЕР®**

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Новосибирск · Ростов-на-Дону · Екатеринбург · Самара
Киев · Харьков · Минск

2004



Попов В. Б.
П58 Паскаль и Дельфи. Самоучитель — СПб.: Питер, 2004. — 544 с.: ил.
ISBN 5-8046-0156-3

Данное пособие представляет собой курс по изучению популярного языка программирования — Паскаль. В нем последовательно излагаются основные принципы структурного и объектно-ориентированного программирования. Наиболее подробно рассматриваются интегрированные среды программирования — Турбо Паскаль и Дельфи.

В каждой главе разбираются примеры рабочих программ. Для самопроверки усвоения теоретического материала вы можете воспользоваться вопросами, приведенными в конце каждой главы. Выполнение заданий по разработке приложений поможет сформировать прочные навыки программирования.

Книга предназначена для учащихся и студентов общеобразовательных, высших и средних учебных заведений и благодаря наличию большого количества детально рассмотренных примеров, вопросов и заданий может быть использована для самообразования.

ББК 32.973-018я7
УДК 681.3.06(075)

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

Краткое содержание

Предисловие	14
Введение	16

Часть I. Основы программирования на языке Turbo Pascal

Глава 1. Алгоритмы и языки программирования	22
Глава 2. Основные элементы языка Pascal	45
Глава 3. Типы данных. Ввод-вывод данных	68
Глава 4. Операторы	99
Глава 5. Процедуры и функции	128
Глава 6. Структурированные типы данных. Строки	158
Глава 7. Массивы	176
Глава 8. Множества и записи	203
Глава 9. Файлы	230
Глава 10. Динамические структуры данных	280

Часть II. Введение в программирование в Delphi

Глава 11. Введение в объектно-ориентированное программирование	298
Глава 12. Интегрированная среда разработки Delphi 6	329
Глава 13. Приложения для обработки строк, массивов и файлов	366
Глава 14. Приложения с мультимедиа	403
Глава 15. Создание простых приложений	443
Приложение А. Глоссарий	474
Приложение Б. Меню интегрированной системы программирования Turbo Pascal 7.0	504
Приложение В. Справочные сведения по среде программирования Delphi 6	522

Содержание

Предисловие 14

Как пользоваться этой книгой 15

Введение 16

Эволюция языков программирования 16

Направления развития языков программирования 17

Интегрированные среды разработки программ 19

Переход к визуальному программированию 19

Часть А. Основы программирования на языке Turbo Pascal

Глава 1. Алгоритмы и языки программирования 22

Язык программирования Pascal 22

Трансляторы 23

Интегрированная среда программирования Turbo Pascal 7.0 23

Назначение и возможности среды программирования Turbo Pascal 23

Основные файлы пакета Turbo Pascal 24

Запуск интегрированной среды программирования Turbo Pascal 25

Справочная система Turbo Pascal 30

Редактор интегрированной среды 31

Ввод текста программы в окне редактора 34

Компиляция программы 34

Создание .exe-файла 34

Исполнение программы 36

Просмотр выполнения программы на экране пользователя 36

Сохранение программы на диске 36

Завершение работы в интегрированной среде программирования 39

Открытие файла с текстом программы 39

Получение справочной информации по редактору 41

Ошибки, обнаруженные при компиляции 42

Содержание

7

Контрольные вопросы и задания	44
Вопросы	44
Задания	44

Глава 2. Основные элементы языка Pascal 45

Алфавит и словарь языка Pascal 45

Символы в Pascal 45

Слова в Pascal 46

Формальные методы описания синтаксических конструкций
языка программирования 48

Идентификаторы 48

Стандартные идентификаторы 49

Пользовательские идентификаторы 49

Константы и переменные 50

Структура Pascal-программы 53

Раздел uses 56

Раздел описания меток 56

Раздел описания констант 57

Раздел описания типов данных 58

Раздел описания переменных 58

Раздел описания процедур и функций 59

Раздел операторов 59

Комментарии 60

Директивы компилятора и управляющие символы 61

Библиотечные модули пользователя 62

Советы по стилю программирования 62

Контрольные вопросы и задания 63

Глава 3. Типы данных. Ввод-вывод данных 68

Общие сведения 68

Перечень типов данных в Turbo Pascal 69

Скалярные типы данных 70

Пользовательские типы 75

Структурированные типы данных 78

Тождественность и совместимость типов 78

Выражения, операции, операнды 80

Арифметические выражения и операции 80

Выражения и операции отношения 83

Логические выражения и операции 84

Приоритет операций 86

Контрольные вопросы и задания 87

Вопросы 87

Задания 87

Ввод-вывод данных	90
Общие сведения	90
Процедуры ввода-вывода	90
Процедура чтения Read	90
Процедура записи Write	92
Контрольные вопросы и задания	96
Вопросы	96
Задания	96

Глава 4. Операторы 99

Общие сведения	99
Простые операторы	99
Оператор присваивания	99
Оператор безусловного перехода (go to)	100
Оператор вызова процедуры	101
Пустой оператор	101
Структурные операторы	101
Составной оператор	101
Условные операторы	102
Операторы повтора	106
Правила пунктуации при записи операторов	115
Получение подсказки по языку программирования	115
Тестирование и отладка программ	117
Контрольные вопросы и задания	122
Вопросы	122
Задания	123

Глава 5. Процедуры и функции 128

Методы программирования	128
Необходимость структуризации в программировании	128
Метод нисходящего проектирования программ	129
Подпрограммы в языке Pascal	130
Стандартные библиотечные модули	131
Процедуры и функции пользователя	134
Процедуры	135
Механизм передачи параметров	140
Рекурсии	150
Нетрадиционное использование подпрограмм	151
Контрольные вопросы и задания	153
Вопросы	153
Задания	155

Глава 6. Структурированные типы данных. Строки . . . 158

Описание строкового типа	159
Строковые выражения	160
Строковые процедуры и функции	162
Упражнения	164
Контрольные вопросы и задания	173
Вопросы	173
Задания	173

Глава 7. Массивы 176

Описание типа «массив»	176
Операции над массивами	179
Операции над элементами массива	179
Сортировка массивов	186
Линейная сортировка (сортировка отбором)	186
Сортировка методом пузырька	188
Метод быстрой сортировки с разделением	189
Бинарный поиск в упорядоченных массивах	192
Контрольные вопросы и задания	198
Вопросы	198
Задания	199

Глава 8. Множества и записи 203

Описание типа «множество»	203
Операции над множествами	204
Контрольные вопросы и задания	212
Вопросы	212
Задания	213
Записи	214
Описание типа «запись»	214
Записи с вариантами	217
Контрольные вопросы и задания	225
Вопросы	225
Задания	226

Глава 9. Файлы 230

Описание файлового типа	230
Средства обработки файлов	231

Текстовые файлы	235
Использование буфера ввода-вывода	237
Типизированные файлы	238
Нетипизированные файлы	251
Некоторые стандартные процедуры и функции обработки файлов модуля Dos	264
Константы	264
Типы файловых записей	265
Переменные	267
Процедуры и функции	267
Процедуры даты и времени	267
Функция статуса диска	268
Процедуры обслуживания прерываний	268
Процедуры обработки файлов	268
Функции обработки файла	268
Процедуры обработки процессов	269
Функции обработки процессов	269
Функции управления средой	269
Дополнительные функции	269
Дополнительные процедуры	269
Контрольные вопросы и задания	274
Вопросы	274
Задания	275

Глава 10. Динамические структуры данных 280

Статические и динамические переменные	280
Указатели	281
Типизированные указатели	282
Нетипизированный указатель (pointer)	284
Доступ к переменной по указателю	284
Управление динамической памятью	285
Процедуры динамического распределения памяти	285
Функции динамического распределения памяти	285
Функции для работы с указателями и адресами	286
Использование указателей для организации связанных списков	287
Упражнения	287
Контрольные вопросы и задания	294
Вопросы	294
Задания	295

Часть II. Введение в программирование в Delphi

Глава 11. Введение в объектно-ориентированное программирование 298

Основные понятия объектно-ориентированного программирования	298
Что такое объекты?	299
Классы объектов	299
Иерархия объектов класса	300
Операции и методы	304
Методы. Инициализация полей объектов	304
Определение методов	306
Использование объектов при визуальном проектировании интерфейса	308
Введение в Object Pascal	309
Нововведения в Object Pascal	309
Общая организация программы в Delphi	310
Области видимости и доступ к объектам, переменным и функциям модуля	314
Приложения Windows	321
Управляемая событиями архитектура Windows-приложения	321
Независимая от аппаратуры графика	321
Многозадачный режим	322
Управление памятью	322
Ресурсы	322
Многодокументный интерфейс	323
Автоматизация реакции системы в виде сообщений	323
Основы единого графического интерфейса	324
Архитектура Windows-программы	326
Контрольные вопросы и задания	328

Глава 12. Интегрированная среда разработки Delphi 6 329

Назначение	329
Общее описание среды	330
Создание, компиляция и отладка простого приложения	335
Создание формы с размещением визуальных компонентов	336
Создание кода — обработчика события	344
Контрольные вопросы и задания	363
Вопросы	363
Задания	364

Глава 13. Приложения для обработки строк, массивов и файлов 366

Обработка строк типа String	366
Создание и обработка линейного массива	369
Линейная сортировка массива	374
Использование компонента StringGrid для представления двумерных массивов	377
Ввод и обработка элементов массива с использованием StringGrid	386
Обработка файлов	389
Контрольные вопросы и задания	401
Вопросы	401
Задания	402

Глава 14. Приложения с мультимедиа 403

Канва и пикселы	403
Рисование на канве по пикселям	404
Рисование пером	407
Рисование кистью	411
Мультипликация движением объекта	424
Воспроизведение звука и видеоклипов	430
Процедуры воспроизведения звуков	431
Функция PlaySound	433
Компонент Animate	436
Универсальный проигрыватель аудио- и видеоинформации MediaPlayer	440
Контрольные вопросы и задания	441
Вопросы	441
Задания	442

Глава 15. Создание простых приложений 443

Консольное приложение	443
Многооконный текстовый редактор	446
Взаимодействие приложения с внешними программами	461
Приложение для работы с базами данных	465
Создание новой таблицы в Database Desktop	465
Создание псевдонима базы данных	467
Заполнение новой таблицы в SQL Explorer	468
Создание формы приложения	469

Приложение А. Глоссарий 474

Приложение Б. Меню интегрированной системы программирования Turbo Pascal 7.0 504

Приложение В. Справочные сведения по среде программирования Delphi 6 522

Система меню	522
Палитра компонентов	527
Новые компоненты	532
Окно редактора кода	532
Инспектор объектов	534
Менеджер проектов	537
Браузер проектов	538
To-Do List: список недоделанных дел	539
Перетаскивание и встраивание окон	539
Управление конфигурацией окон	541

Введение

Эволюция языков программирования

Языки программирования претерпели большие изменения с тех пор, как в 40-х годах XX века началось их использование. Первые языки программирования были очень примитивными и мало чем отличались от формализованных упорядочений двоичных чисел (единиц и нулей), понятных компьютеру. Их называют языками программирования *низкого уровня*. Использование таких языков было крайне неудобно с точки зрения программиста, так как он должен был знать числовые коды всех машинных команд и собственноручно распределять память под команды программы и данные.

Чтобы упростить общение человека с компьютером, были разработаны языки программирования типа Ассемблер, в которых переменные величины стали изображаться символическими именами, а числовые коды операций были заменены на мнемонические (словесные) обозначения, которые легче запомнить. Язык программирования приблизился к человеческому языку, но удалился от языка машинных команд. Поэтому чтобы компьютер мог работать на языке Ассемблера, понадобился транслятор — программа, переводящая текст программы на Ассемблере в эквивалентные машинные команды. Языки типа Ассемблер являются машино-ориентированными, потому что они настроены на структуру машинных команд конкретного компьютера. Разные компьютеры с разными типами процессоров имеют разный Ассемблер.

В 50-х годах XX века, в связи с широким применением компьютеров в различных областях науки и техники возникла серьезная проблема: простые пользователи не могли работать с компьютером из-за сложности языков программирования, а профессиональные программисты были не в состоянии обслужить огромное количество пользователей. Решением данной проблемы явилось создание языков программирования высокого уровня, форма записи программ на которых по сравнению с Ассемблером и машинными языками ближе к традиционной математической форме и разговорному языку. Важным преимуществом языков программирования высокого уровня является машинная независимость, поэтому одна и та же программа на таком языке может

Pascal 7.0. Приложение В раскроет для вас возможности интегрированной среды разработки программ Delphi 6.

Каждая глава содержит большое количество подробно разобранных примеров программ. Все приведенные примеры программ были проверены в среде программирования Turbo Pascal версии 7.0 или среде Delphi 6. Все примеры можно найти в Интернете по адресу: <http://www.vb-popov.narod.ru>.

Как пользоваться этой книгой

Уважаемый читатель, каждая часть книги является самостоятельным разделом, поэтому при изучении учебного материала возможны различные подходы. Начинающим программистам рекомендуем внимательно изучить материал первой части и только затем переходить к последовательному изучению остальных частей книги с выполнением всех практических упражнений. Тем, кто уже знаком с языком программирования Pascal, можно сразу перейти к изучению второй части книги.

В конце каждой из глав приводятся вопросы, которые вы можете использовать для проверки успешности усвоения теоретического материала, и задания для самостоятельной работы.

Пользуясь предметным указателем, вы сможете быстро найти в тексте книги нужное слово.

Очевидно, что одна книга, какой бы толстой она ни была, не может охватить все детали использования среды программирования Turbo Pascal 7.0 и интегрированной среды разработки программ Delphi 6, поэтому в конце книги приводится список литературы и ссылок на информационные ресурсы в Интернете, которые могут расширить ваши знания по программированию.

Выражаю глубокую признательность всем коллегам, которые помогли выходу этой книги. Все замечания и пожелания по поводу данной книги будут приняты автором с благодарностью по адресу: vb-popov@narod.ru.

Предисловие

Уважаемый читатель, эта книга представляет собой курс по изучению одного из популярных языков программирования — Pascal. В ней последовательно излагаются основные принципы структурного программирования, языка программирования Pascal, объектно-ориентированного программирования. Большое внимание в книге уделено изучению использования при разработке программ интегрированных сред программирования Turbo Pascal и Delphi.

В первой части книги вы получите сведения о применении интегрированной среды программирования Turbo Pascal версии 7.0 при разработке программ.

Во второй и третьей главах книги вы познакомитесь с основными элементами языка Pascal и структурой программы на Pascal, получите представление о типах данных, узнаете, как правильно записывать различные выражения и организовывать ввод данных и вывод результатов работы Pascal-программы. В четвертой главе вы научитесь использовать операторы языка Pascal для записи линейных, разветвляющихся и циклических алгоритмов. Изучая пятую главу, вы освоите структурный подход к разработке алгоритмов и программ.

Начиная с шестой главы вы познакомитесь со структурированными типами данных: строками, массивами, множествами, записями и файлами. В десятой главе вы научитесь рационально использовать память компьютера, реализуя динамические переменные.

Вторая часть книги посвящена изучению основ объектно-ориентированного программирования и создания простых приложений Windows в среде программирования Delphi 6. Одиннадцатая глава посвящена основным понятиям объектно-ориентированного программирования. Материал двенадцатой главы поможет вам освоить интегрированную среду разработки Delphi 6. Изучая тринадцатую главу, вы научитесь создавать приложения для обработки строк, массивов и файлов. В четырнадцатой главе вы познакомитесь с созданием мультимедийных приложений. В пятнадцатой главе подробно разобраны примеры создания простых приложений: многооконного текстового редактора, приложений для работы с базами данных, создание консольного приложения и приложения, взаимодействующего с внешними программами.

В приложении А представлен словарь терминов. Материалы приложения Б познакомят вас с меню интегрированной среды программирования Turbo

быть выполнена на компьютерах различных типов, оснащенных соответствующим транслятором. К недостаткам программ, написанных на языках высокого уровня, относятся большой объем занимаемой памяти и более медленное выполнение, чем у программ на машинных языках или языках Ассемблера. Первыми популярными языками высокого уровня, появившимися в 50-х годах XX века, были Fortran, Cobol и Algol.

В 60–70-х годах XX века появилось огромное количество новых языков программирования. В 1965 году появились два новых важных языка. Для обучения программированию был разработан язык, который являлся упрощенной версией Фортрана и получил название Basic (Beginner's All-purpose Symbolic Instruction Code, многоцелевой код символических команд для начинающих). Basic предоставил пользователю разнообразные средства для диалога с компьютером во время выполнения программы. Вторым языком, появившимся в 1965 году, был PL/1 (Programming Language 1, язык программирования 1). При его создании преследовалась цель создать язык, сочетающий в себе лучшие свойства Algol, Cobol и Fortran, и в конечном итоге заменяющий своих предшественников. Однако этого не произошло, в связи с тем что PL/1 не проявил тех преимуществ, которые оправдали бы переход к нему. К тому же большое количество средств и разнообразие операторов ПЛ/1 привели к сложности в его изучении.

В 1971 году профессор Н. Вирт из института информатики Швейцарской высшей политехнической школы в Цюрихе разработал новый язык, получивший название Pascal (в честь математика XVII века Блеза Паскаля). Язык Pascal основан на Алголе и создавался как учебный язык, в нем строго соблюдена структурная линия программирования. В силу своих достоинств Pascal послужил источником для создания многих современных языков программирования, таких как Ada, C и Modula-2.

Язык C первоначально был разработан для компьютеров, использующих операционную систему UNIX. Он является относительно простым языком, в нем нет операций над символьными строками и списками, но, в отличие от Pascal, в нем заложены возможности непосредственного обращения к некоторым машинным командам, к определенным участкам памяти компьютера. Язык C широко используется как инструментальный язык для разработки операционных систем, трансляторов, баз данных, а также других системных и прикладных программ.

Направления развития языков программирования

В развитии языков программирования выделяются два основных направления: процедурное и неперечисленное. В процедурных языках программа явно описывает действия, которые необходимо выполнить, а результат задается

способом получения его при помощи некоторой процедуры — определенной последовательности действий. Основными средствами, применяемыми в этих языках, являются величины (в том числе и табличные), присваивания, циклы, процедуры. При построении процедурной программы необходимо ясно представлять, какие действия и в какой последовательности будут производиться при ее выполнении. Среди процедурных языков можно, в свою очередь, выделить структурные и операционные языки. В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы. В операционных языках для этого используется несколько операций. Широкое распространение получили структурные языки Pascal, C, Ada, PL/1 и операционные языки Fortran, Basic, Focal.

Непроцедурное (декларативное) программирование появилось в начале 70-х годов, но его развитие началось в 80-е годы в связи с проектом по созданию компьютеров пятого поколения, целью которого явилась подготовка почвы для создания интеллектуальных машин. К непроцедурному программированию относятся функциональные и логические языки. В функциональных языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь разбиваются на еще более простые, и т. д. Один из основных элементов в функциональных языках — рекурсия, т. е. вычисление значения функции через значение этой же функции от других элементов. Наиболее распространенными среди функциональных языков являются Lisp и Refal. Lisp, являющийся языком обработки списков, давно и активно применяется в системах искусственного интеллекта. Refal, построенный на алгоритмах Маркова, удобен для обработки текстов и является единственным из распространенных в мире языков программирования, разработанным в нашей стране. Промежуточное положение занимает язык Logo, который содержит средства и процедурного, и функционального программирования. На начальном уровне он похож на классический процедурный язык, а при решении сложных задач обработки данных на первый план выходят функциональные методы.

Функциональная программа, как и процедурная, описывает действия, которые надо совершить для достижения результата (вызов функции — это тоже действие), но ее построение требует скорее математического, чем алгоритмического мышления.

В логической традиции программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого можно задавать вопросы. Машина перебирает известные (заданные в программе) данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается Prolog, хотя он и содержит некоторые средства управления перебором, т. е. процедурные элементы. Построение логической программы вообще не требует алгоритмического мышления. Здесь нет динамики, нет описания действий, программа описывает статические отношения объектов, а динамика находится в механизме перебора и скрыта от программиста.

Можно выделить еще один класс языков программирования — объектно-ориентированные языки сверхвысокого уровня. На таких языках не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно-ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме. Примером такого языка может служить язык программирования визуального общения SmallTalk. Трудно провести четкую границу между системами программирования сверхвысокого уровня и прикладным программным обеспечением. Как те, так и другие системы позволяют работать с ними неквалифицированному пользователю, не являющемуся программистом.

Интегрированные среды разработки программ

Трудоемкость создания сложных компьютерных программ и разнообразие средств, используемых в процессе разработки программы для написания кода, компиляции и отладки, привело в 80-х годах XX века к осознанию необходимости интеграции этих средств. В те годы фирма *Borland International, Inc.* (США) разработала систему Turbo Pascal, которую называют интегрированной (integration — объединение отдельных элементов в единое целое) средой программирования, так как она объединяет в себе возможности ранее разрозненных средств, используемых при разработке программ: редактора текстов, компилятора, компоновщика и отладчика. Часто ее кратко называют IDE (*Integrated Development Environment, интегрированная среда разработки).

Переход к визуальному программированию

В конце XX века стандартом интерфейса пользователя компьютерных программ становится графический интерфейс (GUI).

ПРИМЕЧАНИЕ

GUI (graphical user interface) — графический интерфейс пользователя, использующий окна (windows), значки (icons) и мышь (mouse). Взаимодействие пользователя с программой средствами GUI основано на интуитивно понятных принципах, что обеспечивает продуктивное использование компьютера даже неподготовленным пользователем.

GUI предлагает более сложное и дружелюбное окружение пользователя, чем командно-управляемый интерфейс DOS. Однако при разработке приложений с GUI возникали большие трудности программирования, потому что для отображения объектов интерфейса пользователя (меню, окон, кнопок и т. п.) требовалось написание фрагмента кода программы, создающего и настраи-

вающего объект «по месту». Увидеть же закодированные объекты можно было только в ходе исполнения программы. При таком подходе достижение того, чтобы объекты выглядели и вели себя заданным образом, было утомительным процессом, требующим неоднократных исправлений кода с последующим запуском программы и просмотром полученного результата.

С целью преодоления трудностей на этапе создания интерфейса пользователя в конце XX века широкое распространение получило визуальное программирование. Средства визуального программирования предоставляют программисту возможность наглядным образом конструировать интерфейс, изображая объекты интерфейса на экране на этапе конструирования элементов интерфейса программы с автоматической генерацией соответствующего кода. После того как объект помещен в форму среды визуального программирования, все его атрибуты сразу отображаются в виде кода, который соответствует объекту как единице, исполняемой в ходе работы программы. Благодаря средствам визуальной разработки можно работать с объектами, держа их перед глазами и получая результаты практически сразу. Способность видеть объекты такими, какими они появляются в ходе исполнения программы, снимает необходимость проведения множества операций вручную.

Одним из популярных средств разработки приложений с использованием методов визуального программирования является среда программирования Delphi. Язык, на котором разрабатываются приложения в Delphi, — это Object Pascal. С одной стороны, это прежний Pascal, поскольку все его основные конструкции сохранены. С другой стороны, важнейшая составная часть языка, модель объектов, подверглась коренному преобразованию. Причем Delphi, начиная с 4-й версии, наряду с приложениями, использующими графический пользовательский интерфейс, позволяет создавать консольные приложения.

ПРИМЕЧАНИЕ

Консольные приложения — это особый вид Windows-приложений. С одной стороны, такое приложение имеет полный доступ к функциям Win API, с другой — не имеет графического интерфейса и выполняется в текстовом режиме.

Часть I

Основы программирования на языке Turbo Pascal

Глава 1

Алгоритмы и языки программирования

Язык программирования Pascal

Язык программирования Pascal был разработан в 1968–1971 годах. Никлаусом Виртом. Язык был назван в честь выдающегося французского математика и философа Блеза Паскаля (1623–1662) и первоначально создавался для обучения программированию как систематической дисциплине, однако вскоре он стал широко использоваться в профессиональном программировании.

Популярности языка Pascal среди программистов способствовали следующие факторы.

- Благодаря своей компактности и удачному первоначальному описанию, Pascal оказался достаточно легким для изучения.
- Pascal отражает фундаментальные и наиболее важные концепции (идеи) алгоритмов в очевидной и легко воспринимаемой форме, что предоставляет программисту средства, помогающие проектировать программы.
- Язык позволяет четко реализовать идеи структурного программирования и структурной организации данных.
- Pascal сыграл большую роль в развитии методов аналитического доказательства правильности программ и позволил реально перейти от методов отладки программ к системам автоматической проверки правильности программ.
- Применение языка Pascal позволило значительно поднять планку надежности разрабатываемых программ за счет требований языка к описанию используемых в программе переменных, проверки согласованности программы при компиляции без ее выполнения.
- Использование в Pascal простых и гибких структур управления: ветвлений, циклов.

Трансляторы

Поскольку текст записанной на Pascal программы непонятен компьютеру, требуется перевести его на машинный язык. Такой перевод программы с языка программирования на язык машинных кодов называется *трансляцией* (translation — перевод), а выполняется он специальными программами — *трансляторами*.

Существует три вида трансляторов: интерпретаторы, компиляторы и ассемблеры.

Интерпретатором называется транслятор, производящий пооператорную (покомандную) обработку и выполнение исходного кода программы.

Компилятор преобразует (транслирует) всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и лишь потом исполняется.

Ассемблеры переводят программу, записанную на языке ассемблера (автокода), в программу на машинном языке.

Любой транслятор решает следующие основные задачи:

- анализирует транслируемую программу, в частности определяет, содержит ли она синтаксические ошибки;
- генерирует выходную программу (ее часто называют *объектной*, или *рабочей*) на языке машинных команд (в некоторых случаях транслятор генерирует выходную программу на промежуточном языке, например, на языке ассемблера);
- распределяет память для объектной программы (в простейшем случае это заключается в назначении каждому фрагменту программы, переменным, константам, массивам и другим объектам адресов памяти).

Интегрированная среда программирования Turbo Pascal 7.0

Назначение и возможности среды программирования Turbo Pascal

Для повышения качества и скорости разработки программ в середине 80-х годах была создана система программирования Turbo Pascal. Слово «Турбо» в названии системы программирования — это отражение торговой марки фирмы-разработчика *Borland International, Inc.* (США). Систему программирования Turbo Pascal называют *интегрированной* (integration — объединение отдельных элементов в единое целое) средой программирования, так как она объединяет в себе возможности ранее разрозненных средств, используемых

при разработке программ: редактора текстов, компилятора, компоновщика, отладчика, и при этом обеспечивает программисту великолепные сервисные возможности. Часто ее кратко называют IDE (Integrated Development Environment — интегрированная среда разработки).

Интегрированная среда Turbo Pascal версии 7.0 имеет следующие возможности:

- наличие множества накладываются окон;
- поддержка мыши, меню, диалоговых окон;
- многофайловый редактор;
- расширенные возможности отладки;
- полное сохранение и восстановление среды разработки.

К существенным отличиям версии 7.0 от более ранних версий относятся:

- наличие объектно-ориентированной среды разработки прикладных программ Turbo Vision;
- полные возможности встроенного ассемблера;
- личные поля и методы в объявлении объектов;
- директива расширенного синтаксиса `$X`, позволяющая интерпретировать функции как процедуры (и игнорировать результаты функций);
- адресные ссылки в типизированных константах;
- директивы вызова ближних и дальних процедур;
- редактирование инициализированных данных из объектных файлов;
- расширенные возможности встроенной справочной системы с использованием вырезания и вставки кода примеров для каждой библиотечной процедуры и функции.

Все примеры программ на Pascal, приведенные в данном пособии, подготовлены, откомпилированы и отлажены в интегрированной среде программирования Turbo Pascal версии 7.0.

Основные файлы пакета Turbo Pascal

Если система программирования Turbo Pascal установлена на диске C: в каталоге C:\TP, то в каталоге C:\TP\BIN находятся следующие основные файлы Turbo Pascal:

TURBO.EXE — интегрированная среда программирования;

TURBO.HLP — файл, содержащий данные для оперативной подсказки;

TURBO.TP — файл конфигурации системы;

TURBO.TPL — библиотека стандартных модулей Turbo Pascal.

В каталоге C:\TP\BGI находятся файлы, необходимые для работы в графическом режиме: GRAPH.TPU — модуль с графическими процедурами и функциями

Turbo Pascal, несколько файлов с расширением .BGI — драйверы различных типов видеосистем, несколько файлов с расширением .CHR, содержащих векторные шрифты.

В каталоге C:\TP\UNITS располагаются различные модули с библиотеками процедур и функций.

Запуск интегрированной среды программирования Turbo Pascal

Для запуска интегрированной среды программирования следует перейти в каталог с Turbo Pascal и ввести команду TURBO.EXE. После запуска программы экран компьютера будет иметь вид, показанный на рис. 1.1.

На экране отображаются три компонента интегрированной среды программирования: полоса меню в верхней части, область окна в центре и строка состояния внизу. Рассмотрим каждый из компонентов.

Полоса меню и подменю. Полоса меню является основным средством доступа к командам меню. Она становится невидимой только в тот момент, когда производится просмотр вывода программы. Если полоса меню активна, то заголовок меню выделен; это текущее выбранное меню. Если за командой меню следует знак многоточия (...), то выбор команды приводит к отображению диалогового окна. Если за командой следует стрелка (▶), то это меню содержит подменю. Команда без знака многоточия или стрелки указывает, что ее выбор приведет к выполнению какого-либо действия.

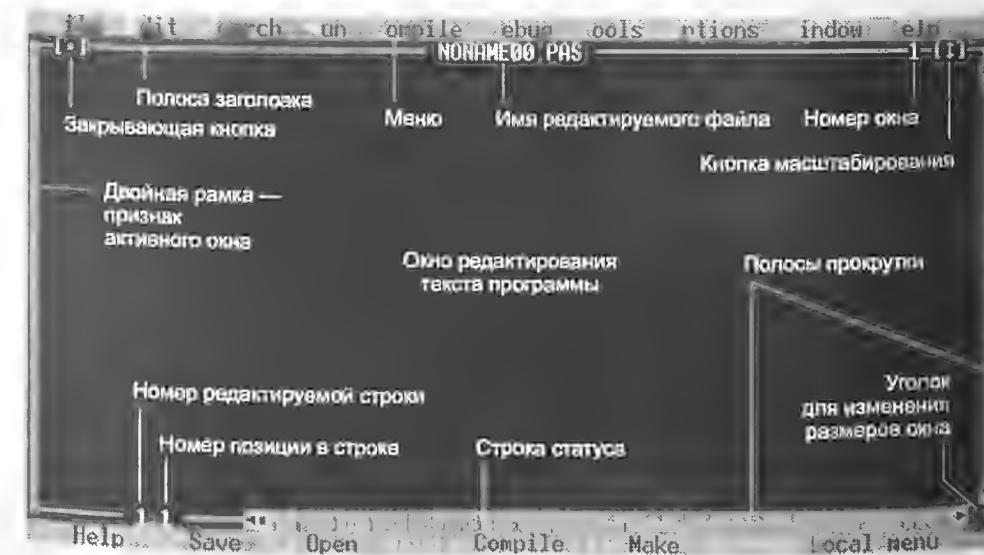


Рис. 1.1. Элементы окна редактирования интегрированной среды программирования Turbo Pascal 7.0

Строка состояния (статуса). Строка состояния отображается в нижней строке экрана и выполняет следующие функции:

- напоминает основные сочетания клавиш, допустимых в этот момент в активном окне;
- предоставляет возможность быстрого выполнения действий путем указания их мышью в строке состояния вместо выбора команд из меню или нажатия последовательности клавиш;
- содержит информацию о выполняемой в данный момент функции. Например, когда производится сохранение редактируемого файла, в строке состояния отображается сообщение Saving filename...;
- предлагает краткие советы по выбранной команде меню и элементам диалогового окна.

При смене окна или изменении характера деятельности информация в строке состояния немедленно меняется. Одна из наиболее характерных строк состояния — та, которая отображается в момент редактирования текста программы в окне редактора.

Использование клавиатуры для выбора команды меню

Для выбора команды меню с помощью клавиатуры необходимо произвести следующие действия.

1. Нажать клавишу F10. Это делает полосу меню активной.
2. Чтобы выбрать меню, следует воспользоваться клавишами со стрелками, а затем нажать Enter. Можно просто нажать выделенную букву заголовка меню. Например, можно нажать E в полосе меню, чтобы открыть меню Edit. Чтобы прервать выполняемое действие, следует нажать клавишу Esc.
3. Теперь следует снова воспользоваться клавишами со стрелками для выбора требуемых команд и нажать Enter. Для быстроты можно нажать выделенную букву команды, как только появилось меню. В этот момент Turbo Pascal выполняет команду, отображает диалоговое окно или показывает другое меню.

Использование мыши для выбора команд меню

Для выбора команды меню с помощью мыши следует выполнить следующие действия.

1. Щелкнуть мышью заголовок требуемого меню.
2. Щелкнуть мышью требуемую команду. Можно также перетащить мышью заголовок меню к команде меню и отпустить кнопку мыши на требуемой команде. (Если вы изменили свое решение, выйдите из меню; ни одна команда не будет выбрана).

Заметим, что некоторые команды меню являются недоступными, когда их выбор не имеет смысла. Однако можно щелкнуть мышью на недоступной команде, чтобы получить по ней подсказку.

ПРИМЕЧАНИЕ

В Turbo Pascal используется только левая кнопка мыши. Однако вы можете самостоятельно настроить правую кнопку, а также сделать другие настройки для мыши.

Быстрые способы выбора команд меню

Turbo Pascal предлагает несколько быстрых способов выбора команд меню. Например, можно преобразовать двухшаговый процесс в одношаговый, проведя мышью от заголовка меню вниз к командам меню с отпусанием кнопки мыши при выборе требуемой команды.

Для клавиатуры можно использовать несколько быстрых методов (или «горячих клавиш») для доступа к полосе меню и выбора команд. Быстрые методы для диалоговых окон работают так же, как и для меню. При перемещении от окна ввода к группе кнопок или окон следует держать нажатой клавишу Alt при нажатии выделенной буквы. Многим элементам меню назначены соответствующие «горячие клавиши», которые немедленно активизируют команду или открывают диалоговое окно. Можно также щелкнуть мышью по имени команды в строке состояния.

В табл. 1.1 перечислены сочетания клавиш, наиболее часто используемые в Turbo Pascal.

Таблица 1.1. Общие клавиатурные комбинации

Клавиша(и)	Элемент меню	Функция
F1	Help	Вывод на экран окна подсказки
F2	File ▶ Save	Сохранение файла, находящегося в активном окне редактора
F3	File ▶ Open	Отображает диалоговое окно и предлагает возможность открыть файл
F4	Run ▶ Go to Cursor	Запуск программы до строки, на которой стоит курсор
F5	Window ▶ Zoom	Масштабирование активного окна
F6	Window ▶ Next	Переход к следующему открытому окну
F7	Run ▶ Trace Into	Запуск программы в режиме отладки с заходом внутрь процедур
F8	Run ▶ Step Over	Запуск программы в режиме отладки, минуя вызовы процедур
F9	Compile ▶ Make	Запуск Make для текущего окна
F10	(none)	Возврат в полосу меню

Окна Turbo Pascal

Почти все, что вы видите и делаете в среде Turbo Pascal, происходит в окнах. Интегрированная среда программирования Turbo Pascal 7.0 позволяет иметь любое количество открытых окон (если есть свободная память), но в любой момент времени может быть активным только одно окно. *Активное окно* —

это окно, с которым в данный момент времени производится работа. Любая выбранная команда или введенный текст относятся только к активному окну. (Если один и тот же файл открыт в нескольких окнах, действие будет применяться к файлу везде.)

ПРИМЕЧАНИЕ

Можно увеличить число окон, которые потенциально могут быть открыты, путем увеличения размера динамической памяти. Для этого используется команда (Options ► Environment).

Существует несколько типов окон, но большинство из них имеют следующие элементы:

- полоса заголовка;
- закрывающая кнопка;
- полосы прокрутки;
- уголок для изменения размеров окна;
- кнопка масштабирования;
- номер окна.

Расположение этих элементов показано на примере окна редактирования текста программы на рис. 1.1.

Быстрый выбор какого-либо пункта меню выполняется клавишами, показанными в табл. 1.2.

Таблица 1.2. Клавиатурные комбинации меню

Клавиши	Элемент меню	Функция
Alt+ПРОБЕЛ	меню —	Переход в меню (System)
Alt+C	меню Compile	Переход в меню Compile
Alt+D	меню Debug	Переход в меню Debug
Alt+E	меню Edit	Переход в меню Edit
Alt+F	меню File	Переход в меню File
Alt+H	меню Help	Переход в меню Help
Alt+O	меню Options	Переход в меню Options
Alt+R	меню Run	Переход в меню Run
Alt+S	меню Search	Переход в меню Search
Alt+W	меню Window	Переход в меню Window
Alt+X	File ► Exit	Завершение работы и выход в DOS

Активное окно можно различить по двойной рамке. Оно всегда имеет закрывающую кнопку, кнопку масштабирования, кнопки перемещения и уголок изменения размеров. Если окна перекрываются, то активное окно всегда располагается поверх остальных (на переднем плане).

ПРИМЕЧАНИЕ

Если вы изменили файл в окне редактирования, то в строке состояния слева от номеров строки и столбца появляется знак звездочки (*).

Закрывающая кнопка окна — это кнопка в левом верхнем углу. Нажав эту кнопку, можно быстро закрыть окно. Можно также выбрать Window ► Close или нажать клавиши Alt+F3. Окно справочной информации рассматривается как временное и может быть закрыто посредством нажатия клавиши Esc.

Полоса заголовка — находящаяся в верхней части окна горизонтальная строка, содержащая имя и номер окна. Для масштабирования окна следует дважды щелкнуть мышью на полосе заголовка. Можно также переместить окно путем перетаскивания строки заголовка. Каждое открытое окно в Turbo Pascal имеет номер, находящийся в верхнем правом углу. Нажатие клавиш Alt+0 позволяет получить список всех открытых окон. Можно сделать окно активным посредством нажатия клавиши Alt в комбинации с номером окна. Например, если окно справочной информации имеет номер 3, но оно скрыто другими окнами, то для быстрого вынесения его на передний план можно нажать одновременно клавиши Alt и 3.

Кнопка масштабирования окна находится в верхнем правом углу окна. Если значок в этом углу изображает стрелку вверх, то щелчок мышью на этой стрелке позволяет увеличить окно до максимально возможного размера. Если значок представляет собой двунаправленную стрелку, то окно уже имеет максимальный размер, и щелчок на стрелке вернет окно к его предыдущему размеру. Чтобы осуществить масштабирование окна с помощью клавиатуры, следует выбрать Window ► Zoom или нажать клавишу F5.

СОВЕТ

Дважды щелкните левой кнопкой мыши на заголовке окна для его масштабирования или восстановления.

Уголок изменения размеров находится в нижнем правом углу окна. Перетаскивание уголка позволяет увеличить или уменьшить размеры окна. Для изменения размеров окна с помощью клавиатуры следует выбрать команду Window ► Size ► Move или одновременно нажать клавиши Ctrl и F5.

Управление окнами в интегрированной среде программирования Turbo Pascal выполняется следующим образом (табл. 1.3).

Таблица 1.3. Управление окнами в среде программирования Turbo Pascal

Операция над окном	Способы выполнения
Открыть окно Edit	Выберите File ► Open для открытия файла и укажите его в диалоговом окне или нажмите клавишу F3
Открыть другие окна	Выберите требуемое окно из меню Window
Закрыть окно	Выберите Close из меню Window (или нажмите клавиши Alt+F3) или щелкните мышью на закрывающей кнопке окна

продолжение

Таблица 1.3 (продолжение)

Операция над окном	Способы выполнения
Активизировать окно	Щелкните мышью в любом месте окна или нажмите клавишу Alt и номер окна (в верхнем правом углу окна), или выберите Window ► List, или нажмите Alt+0 и выберите окно из списка, или выберите Window ► Next или F6, чтобы сделать активным следующее окно. Или нажмите клавиши Alt+F6 для активизации предыдущего окна
Передвинуть активное окно	Перетащите заголовок окна или нажмите клавиши Ctrl+F5 Window ► Size ► Move и используйте клавиши со стрелками для перемещения окна, а затем нажмите Enter
Изменить размер активного окна	Перетащите уголок для изменения размера или любой угол, либо выберите Window ► Size ► Move и нажимайте Shift одновременно с клавишами со стрелками, а затем нажмите Enter. Более быстрый способ состоит в нажатии Ctrl+F5 и одновременном использовании Shift и клавиш со стрелками
Масштабировать активное окно	Щелкните мышью на кнопке масштабирования в верхнем правом углу окна, или дважды щелкните на заголовке окна или выберите Window ► Zoom, либо нажмите клавишу F5

Быстрый способ управления окнами выполняется клавишами, показанными в табл. 1.4.

Таблица 1.4. Клавиатурные комбинации управления окнами

Клавиша(и)	Элемент меню	Функция
Alt+#	(none)	Отображает окно, где # — номер окна
Alt+0	Window ► List	Отображает список открытых окон
Alt+F3	Window ► Close	Закрывает активное окно
Alt+F5	Window ► User Screen	Отображает экран пользователя
Shift+F6	Window ► Previous	Перемещение назад через все открытые окна
F5	Window ► Zoom	Увеличивает (уменьшает) активное окно
F6	Window ► Next	Перемещение вперед через все активные окна
Ctrl+F5	Window ► Size ► Move	Изменяет размер или положение активного окна

Справочная система Turbo Pascal

Как было показано выше, интегрированная среда программирования Turbo Pascal 7.0 предлагает расширенные возможности встроенной справочной системы, которая позволяет программисту не только получить контекстно-ориентированную справочную информацию, но и осуществлять копирование и вставку кода примеров для каждой библиотечной процедуры и функции в текст своей программы, возвращаться назад к другим экранам подсказки (клавиши Alt+F1), пользоваться подсказкой по справочной информации (клавиша F1, если вы уже находитесь в системе справочной информации).

ПРИМЕЧАНИЕ

Справочная система Turbo Pascal получила название контекстно-ориентированной за возможность получения справочной информации, связанной с текущим состоянием среды программирования, по указанному элементу языка программирования. Например, для получения справочной информации о любом пункте меню интегрированной среды программирования следует активизировать этот пункт и нажать клавишу F1; для получения справки по элементу языка в окне редактирования (оператору, функции и т. п.) следует установить курсор на нужном элементе и нажать клавиши Ctrl+F1.

Для получения справочной информации (за исключением случаев, когда управление переходит к вашей программе) нужно нажать клавишу F1 или щелкнуть мышью на соответствующем пункте меню Help. Меню Help (клавиши Alt+N) содержит подробное оглавление системы справочной информации и возможности поиска (Ctrl+F1). Любой экран справочной информации может содержать одно ключевое слово или несколько ключевых слов (выделенных элементов), по которым можно получить дополнительную справочную информацию.

Использование клавиш для получения справочной информации представлено в табл. 1.5.

Таблица 1.5. Клавиатурные комбинации для получения справочной информации

Клавиша(и)	Элемент меню	Функция
F1	Help ► Contents	Открывает экран контекстно-ориентированной справочной информации
F1, F1	Help ► Help on Help	Вызывает справочную информацию по справочной информации (нужно нажать только клавишу F1, если вы уже находитесь в системе справочной информации)
Shift+F1	Help ► Index	Вызывает оглавление справочной информации
Alt+F1	Help ► Previous Topic	Показывает предыдущий экран справочной информации
Ctrl+F1	Help ► Topic Search	Вызывает справочную информацию по языку; действует только в редакторе

Редактор интегрированной среды

Составной частью интегрированной среды разработки программ является редактор Turbo Pascal, предлагающий следующие возможности:

- поддержка мыши;
- поддержка больших файлов (до 1 Мбайта; ограничение в 2 Мбайта для всех комбинаций редактора);
- Shift + клавиши со стрелками — для выбора текста;
- окна редактора, которые можно перемещать и изменять в размере;
- возможность открывать несколько файлов одновременно;
- многочисленные окна, позволяющие иметь несколько представлений одного и того же файла или разных файлов;

- удобный макроязык, позволяющий создавать свои собственные команды редактирования;
- копирование текста или примеров из окна справочной информации;
- редактируемый буфер обмена, допускающий копирование, вставку и передачу данных между окнами.

Для управления редактором используются клавиши, описанные в табл. 1.6.

Таблица 1.6. Клавиатурные комбинации редактирования

Клавиша(и)	Элемент меню	Функция
Shift+стрелки	(none)	Помечает фрагмент текста в активном окне редактирования
Ctrl+Del	Edit ► Clear	Удаляет выбранный текст из окна и не помещает его в буфер обмена
Ctrl+Ins	Edit ► Copy	Копирует выбранный текст в буфер обмена
Shift+Del	Edit ► Cut	Помещает выбранный текст в буфер обмена и удаляет его
Shift+Ins	Edit ► Paste	Помещает текст из буфера обмена в активное окно
Ctrl+L	Search ► Search Again	Повторяет последнюю команду Find или Replace
F2	File ► Save	Сохраняет файл в активном окне редактора
F3	File ► Open	Позволяет открыть файл

Интегрированная система программирования Turbo Pascal включает в себя средства для трансляции программ и их отладки (компилятор, компоновщик, отладчик). Быстрое управление этими средствами с помощью клавиатурных комбинаций показано в табл. 1.7.

Таблица 1.7. Клавиатурные комбинации компиляции, запуска и отладки программ

Клавиша(и)	Элемент меню	Функция
Alt+F9	Compile ► Compile	Компилирует последний файл в окне редактора
Ctrl+F2	Run ► Program Reset	Переустанавливает выполняемую программу
Ctrl+F4	Debug ► Evaluate ► Modify	Вычисляет выражение
Ctrl+F7	Debug ► Add Watch	Добавляет выражение для просмотра
Ctrl+F9	Run ► Run	Запускает программу
F4	Run ► Go To Cursor	Запускает программу до позиции курсора
F7	Run ► Trace Into	Выполнение с трассировкой процедур
F8	Run ► Step Over	Выполнение без трассировки процедур
F9	Compile ► Make	Выполняет операцию Make

Остальные возможности интегрированной системы программирования Turbo Pascal подробно описаны в приложении Б.

Упражнение 1. Создадим программу вычисления суммы двух целых чисел. Можно предложить следующий сценарий взаимодействия человека и компьютера при решении данной задачи.

Компьютер запрашивает у человека значение первого целого числа, считывает его и записывает в память под именем А, затем запрашивает значение второго целого числа, считывает его и записывает в память под именем В. После этого компьютер выполняет сложение чисел А и В, записывает результат в память под именем Summa, выводит на экран сообщение «Сумма чисел =» и печатает значение величины Summa.

Запись данного алгоритма на Pascal может быть представлена в виде следующей программы:

```
{Учебная программа Вычисление суммы двух целых чисел}
program Tutor1;           {Заголовок программы}
var                       {Описание раздела переменных}
  A,B, Summa : integer;   {Переменные A,B,Summa – целые}
begin                     {Начало программы}
  Write('Введите значение целого числа A >'); {Вывод запроса на экран}
  Readln(A);              {Ввод значения A с клавиатуры}
  Write('Введите значение целого числа B >');
  Readln(B);
  Summa := A + B;          {Вычисление переменной Summa}
  Write('Сумма чисел ',A,' и ',B,' = ',Summa); {Вывод ответа}
end.                       {Конец программы}
```

Просмотрите текст программы, обращая внимание на ее структуру.

ПРИМЕЧАНИЕ

- В данной программе использованы следующие зарезервированные слова языка Pascal (слова, за которыми закреплено строго определенное значение):
 - **program** — заголовок программы (определяет ее название и список параметров). Заголовок является декоративным и не оказывает влияния на саму программу;
 - **var** — начало объявления переменных (связывает идентификатор — имя переменной и ее тип с местом в памяти, где хранится ее значение);
 - **integer** — указание, что переменные А, В, Summa — целые числа, т. е. они могут принимать целочисленные значения, такие как 2, 3, 0, 287, 21, 0, 32, 287 и другие, в интервале [-32768, 32767];
 - **begin** — начало тела программы;
 - **end** — конец тела программы;
 - **Write('Текст')** — инструкция компьютеру о выводе на экран сообщения 'Текст' (обратите внимание на то, что текст справа и слева ограничен символом ' — апостроф);
 - **Readln(A)** — инструкция компьютеру о считывании значения переменной А с клавиатуры.
 - Для вычисления суммы чисел А и В в программе использована запись инструкции выполнения вычислений присваивания суммы чисел А и В переменной Summa (присваивание записывается как «:=»): Summa := A + B.
 - Каждая строка программы завершается знаком «;», в конце программы ставится «.».
- Пояснения к программе, не влияющие на ее выполнение, записываются в фигурных скобках {комментарий} или в круглых скобках со звездочкой (* пояснение *).

Ввод текста программы в окне редактора

Для запуска среды программирования Turbo Pascal следует ввести команду turbo и нажать Enter или щелкнуть на соответствующем значке. После запуска программы на экране появляется окно редактирования, в которое следует ввести текст программы. Команды редактора перечислены в табл. 1.6. Для использования дополнительных возможностей следует вызвать локальное меню нажатием клавиш Alt+F10. Текст программы в окне редактирования может выглядеть, как показано на рис. 1.2.

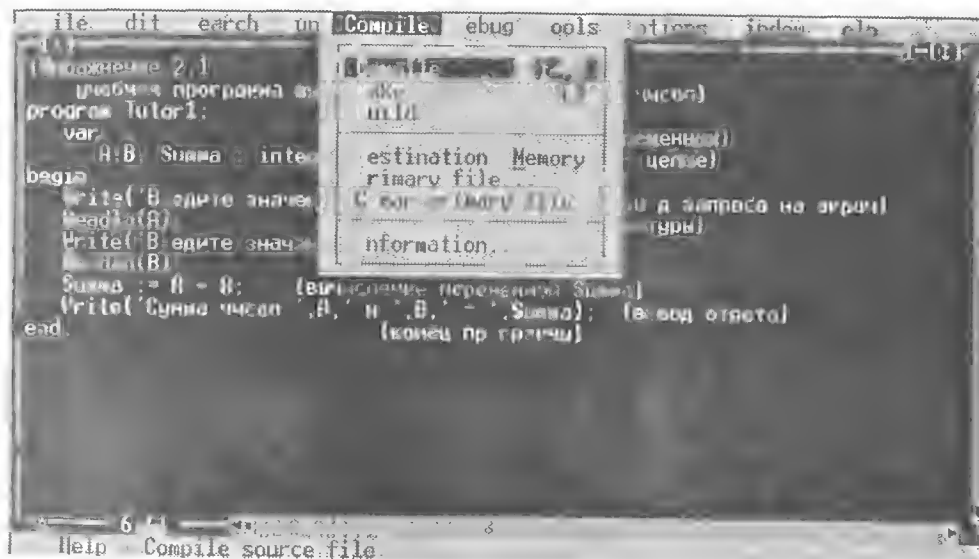


Рис. 1.2. Окно редактирования с текстом программы из упражнения 1

Компиляция программы

Чтобы выполнить компиляцию программы, выберите в меню Compile команду Compile, как показано на рис. 1.2, или нажмите Alt+F9 (см. табл. 1.7). Если вы ввели текст правильно, то на экран будет выведено сообщение об успешном завершении компиляции, показанное на рис. 1.3.

В ответ на сообщение «Compile successful» (успешная компиляция) нажмите любую клавишу.

Создание .exe-файла

Если требуется записать программу на диск в виде исполняемого файла (с расширением .exe), то следует выбрать в меню Compile команду Destination (назначение), и если справа от нее стоит слово Memory (память), указывающее,

что исполняемый код будет храниться в памяти, нажмите клавишу Enter или щелкните левой кнопкой мыши (при этом установка назначения изменится на Disk (диск), как показано на рис. 1.4).

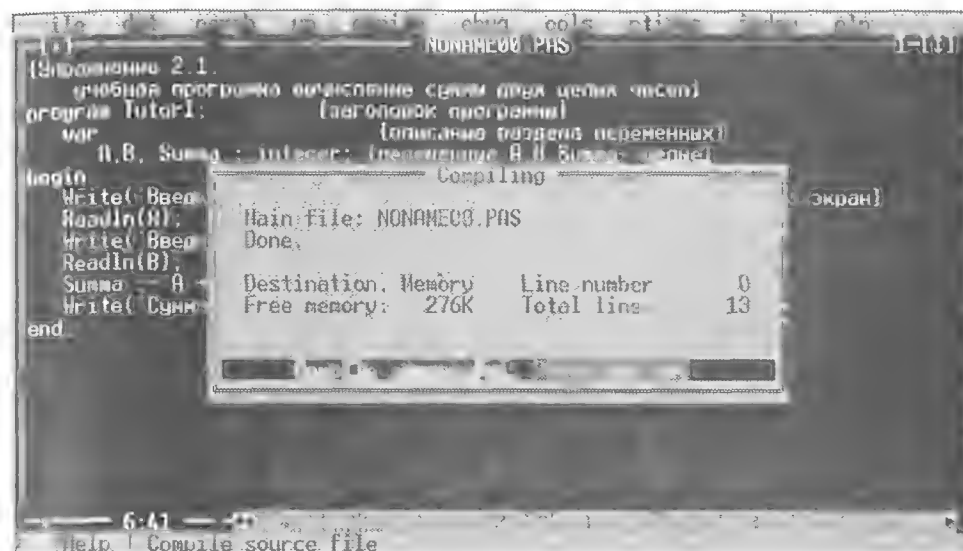


Рис. 1.3. Окно сообщения о результате компиляции

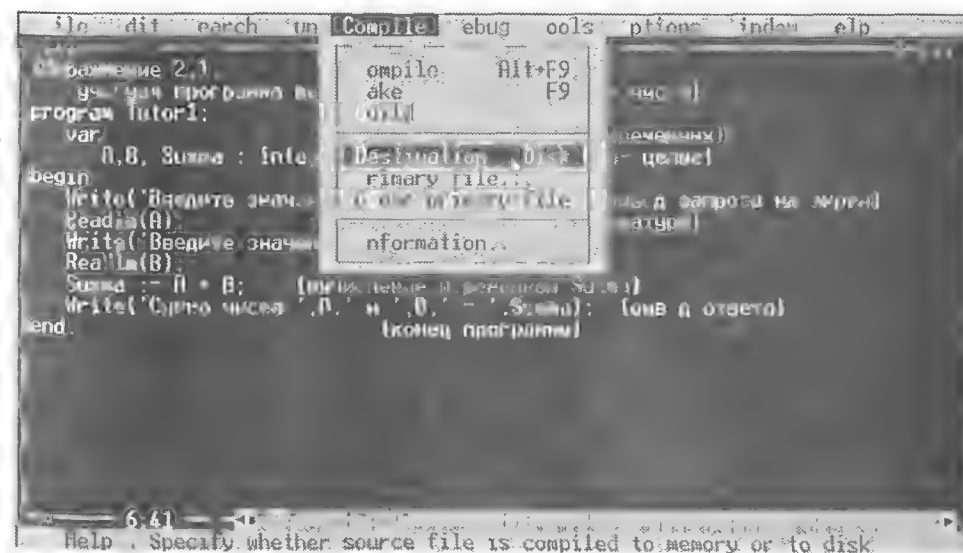


Рис. 1.4. Окно с установкой Destination в положение Disk

Если для опции Destination установлено значение Disk, что указывает на запись исполняемого кода на диск в виде файла с расширением .exe, то следует

перейти к опции Make этого пункта меню или нажать клавишу F9. При этом выполняется создание .exe-файла на диске.

СОВЕТ

Подробнее об использовании меню Compile главного меню интегрированной среды программирования Turbo Pascal 7.0 читайте в приложении Б.

Исполнение программы

В ответ на сообщение «Compile successful» (успешная компиляция) следует нажать любую клавишу, а затем запустить программу на исполнение клавишами Ctrl+F9 (см. табл. 1.7). После этого раскроется экран пользователя и на нем появится сообщение:

Введите значение целого числа A >

На этот запрос введите целое число (например, 3) и нажмите Enter.

Появится следующее сообщение:

Введите значение целого числа B >

На этот запрос введите целое число (например, 4) и нажмите Enter.

После этого будет выполнен расчет суммы с выводом результата на экран, и среда программирования активизирует окно редактирования так быстро, что вы не успеете увидеть результат.

Просмотр выполнения программы на экране пользователя

Чтобы увидеть результат выполнения программы на экране пользователя, следует выбрать Window ► User Screen или нажать Alt+F5. Для значений переменных A = 3, B = 4 экран компьютера будет выглядеть, как показано на рис. 1.5.

Изучите информацию, выведенную программой на экран пользователя, сопоставьте ее с ожидаемой и оцените правильность выполнения программы. Для возврата в среду Turbo Pascal снова нажмите клавиши Alt+F5.

Сохранение программы на диске

Пока файл текста первой программы имеет имя NONAME00, т. е. ему не присвоено конкретного имени. Сохраните текст программы на диске. Имя файла должно отражать назначение программы и быть уникальным.

ВНИМАНИЕ

Имя программы задается в соответствии с правилами DOS, не более 8 латинских символов.

Запишите программу на диск под именем Tutor1, для чего клавишами Alt+F перейдите в меню File (см. табл. 1.2), выберите пункт Save as... (записать под новым именем), как показано на рис. 1.6.

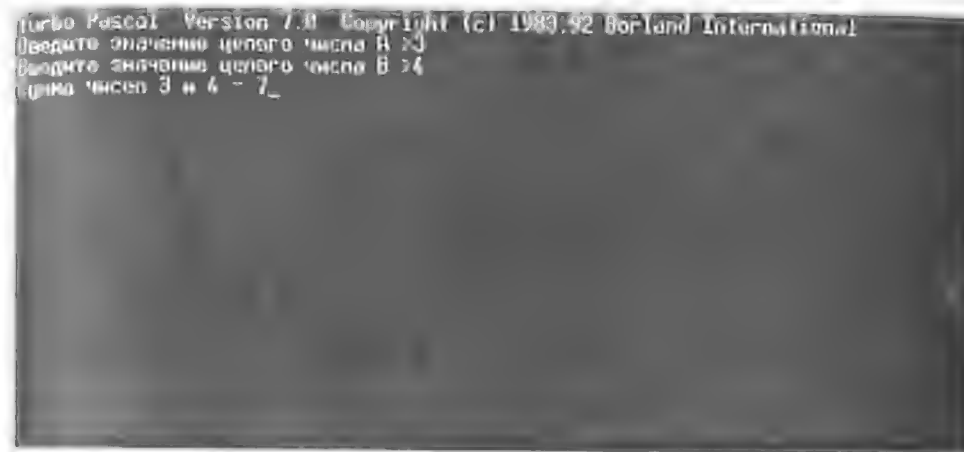


Рис. 1.5. Экран пользователя после выполнения программы из упражнения 1

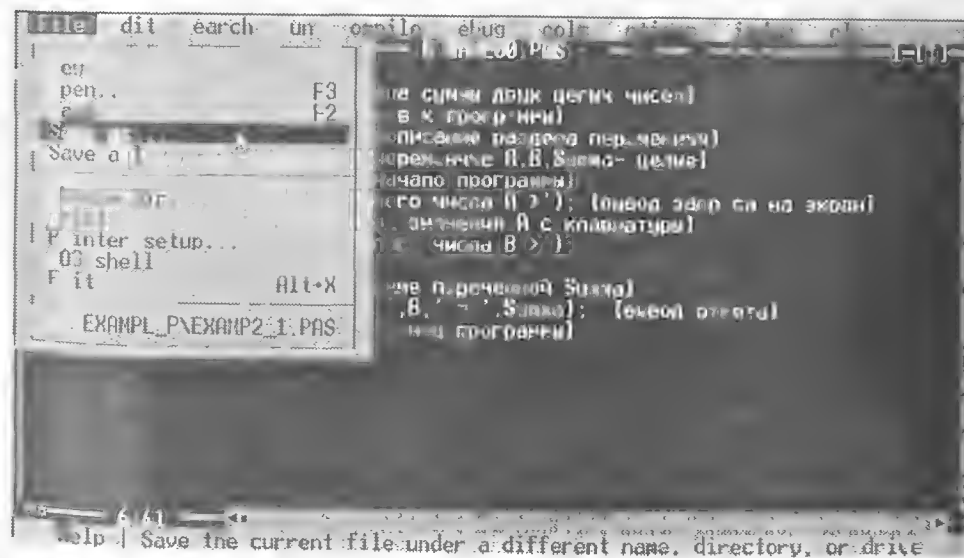


Рис. 1.6. Выбор пункта меню File ► Save as... для записи файла под новым именем

После этого на экране появляется диалоговое окно Save File As, как показано на рис. 1.7. В окне ввода задайте имя программы Tutor1, нажимая Tab или Shift+Tab для перехода от одного элемента к другому (активный элемент выделяется), перейдите к окну списка Files и установите в качестве текущего каталог C:\TP\, в который требуется записать файл текста программы.

После того как залано имя файла и выбран каталог, в который он будет записан, следует нажать кнопку OK. Если вы передумали, нажмите кнопку Cancel или клавишу Esc.



Рис. 1.7. Диалоговое окно Save File As

В случае затруднений следует нажать кнопку Help, и на экран будет выведена подсказка о диалоговом окне Save File As, как показано на рис. 1.8.

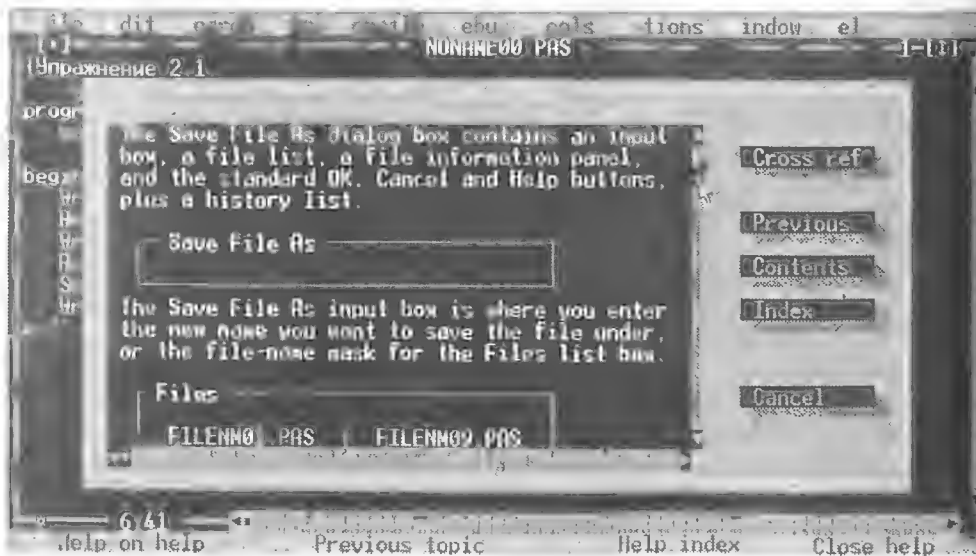


Рис. 1.8. Окно справочной информации о диалоговом окне Save File As

Для использования дополнительных возможностей локального меню следует нажать Alt+F10. Подсказку можно отменить, нажав клавишу Esc.

Если вы еще не записали файл на диск, нажмите кнопку OK, и файл будет записан. Теперь в строке заголовка окна редактирования этого файла отображаются имя файла и каталог, в котором он находится C:\TP\TUTOR1.PAS, как показано на рис. 1.9.

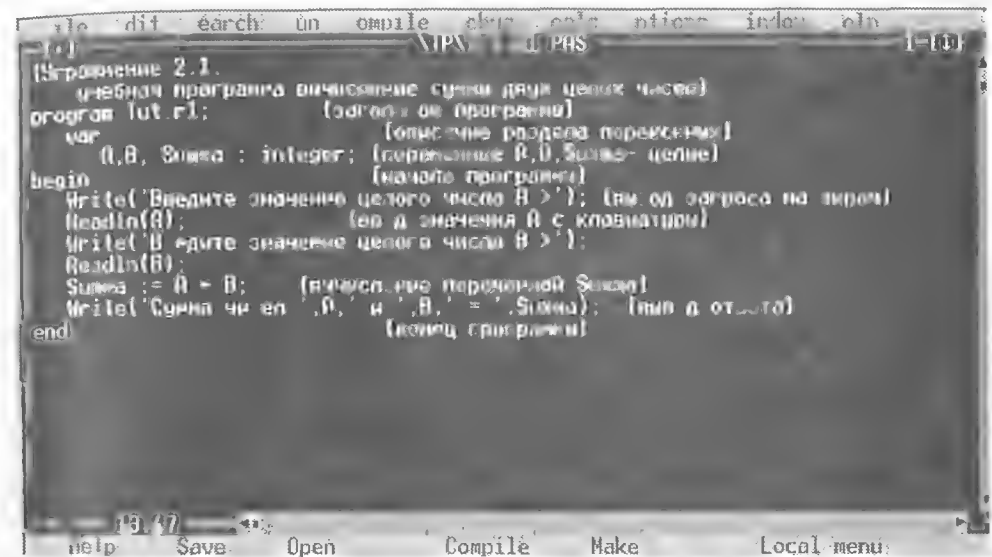


Рис. 1.9. Вид окна редактирования после записи файла TUTOR1 на диск

Завершение работы в интегрированной среде программирования

Завершите работу интегрированной среды программирования Turbo Pascal, для чего клавишами Alt+F перейдите в меню File, в этом меню выберите пункт Exit или нажмите Alt+X (см. табл. 1.2).

Упражнение 2. Измените программу так, чтобы она вычисляла не только сумму, но и разность двух целых чисел ($A - B$). Сценарий взаимодействия человека и компьютера при решении данной задачи аналогичен сценарию в первой задаче (упражнение 1), а алгоритм дополнен вычислением разности двух целых чисел и выводом результата ее вычисления на экран.

Открытие файла с текстом программы

Запустите интегрированную среду программирования Turbo Pascal и прочитайте с диска файл с текстом первой программы TUTOR1.PAS, для чего клавишами Alt+F перейдите в меню File, выберите пункт Open или нажмите клавишу F3. На экране появится окно выбора файла из списка, как показано на рис. 1.10.

Нажимая клавиши Tab или Shift+Tab для перехода от одного элемента к другому (активный элемент выделяется), перейдите к окну списка Files и установите в качестве текущего каталог C:\TP\TUTOR. из которого требуется считать файл с текстом программы. Установите курсор на имени файла TUTOR1.PAS, после этого нажатием клавиши Tab выберите кнопку Open. Если вы передумали, то выберите кнопку Cancel или нажмните клавишу Esc.

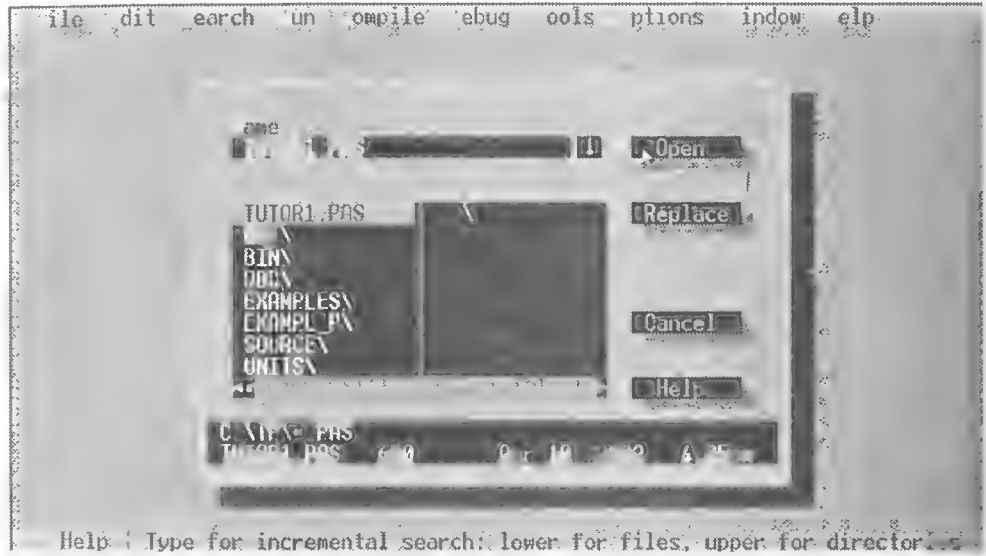


Рис. 1.10. Окно редактирования с панелью выбора файла из списка

В случае возникновения затруднений, нажимая клавишу Tab, выберите кнопку Help и просмотрите подсказку. Для отказа от подсказки нажмните клавишу Esc. Нажатием клавиши Tab выберите кнопку Open. Файл TUTOR1.PAS будет открыт в окне редактирования.

Отредактируйте текст программы, изменив его следующим образом:

```
{Учебная программа Вычисление суммы и разности двух целых чисел}
program Tutor2;                {Заголовок программы}
var                             {Описание раздела переменных}
  A,B, Summa,Raznost : Integer;
                                {Переменные A, B, Summa, Raznost -
                                {целые числа}
begin                           {Начало программы}
  Write('Введите значение целого числа A >'); {Вывод запроса на экран}
  Readln(A);                    {Ввод значения A с клавиатуры}
  Write('Введите значение целого числа B >');
  Readln(B);
  Summa := A + B;               {Вычисление переменной Summa}
  Raznost := A - B;             {Вычисление разности чисел}
```

```
WriteLn('Сумма чисел ' A, ' и ' B, ' = ' Summa);
                                {Вывод ответа с переводом курсора
                                {на следующую строку}
Write('Разность чисел ' A, ' и ' B, ' = ' Raznost);
end                             {Конец программы}
```

Получение справочной информации по редактору

Для получения справочной информации по операциям редактирования посмотрите табл. 1.6 или клавишей F1 вызовите экран подсказки; нажимая клавишу PageDown, перейдите к перечню подсказок о функциях редактирования, как показано на рис. 1.11.



Рис. 1.11. Окно справочной системы со списком функций редактирования

Нажимая клавишу Tab, выберите пункт Using the editor, затем, выбирая: Cursor Movement Commands — перемещение курсора, Insert & Delete Commands — команды вставки и удаления, Block Commands — операции с блоками текста, Miscellaneous Commands — другие команды и нажимая Alt+F1 для возврата к предыдущему экрану помощи, найдите нужную справочную информацию. Чтобы закрыть окно подсказки, нажмните Esc. После завершения редактирования текста программы выполните ее компиляцию, для чего нажмните Alt+F9. Если программа отредактирована без ошибок, запустите ее на выполнение клавишами Ctrl+F9.

Ошибки, обнаруженные при компиляции

Часто в результате компиляции на экран выводятся сообщения об ошибках. Это объясняется тем, что язык программирования Pascal имеет грамматические правила, которые нужно выполнять. Однако в отличие от английского или русского языка в программе на Pascal недопустимо использование сленга или неправильного синтаксиса — компилятор должен всегда однозначно понимать, что вы хотите сказать. Наиболее частыми ошибками начинающих программистов на Pascal бывают:

Unknown identifier (неизвестный идентификатор)

или ';' expected (пропущен символ ';') и другие.

Pascal требует, чтобы вы объявили все переменные, типы данных, константы и подпрограммы, т. е. все идентификаторы, перед их использованием. Если вы обратитесь к необъявленному идентификатору или пропустите его, то возникнет ошибка. Другой частой ошибкой является несоответствие пар `begin..end`, присваивание несовместимым типам данных (например, присваивание действительного значения целой переменной), несоответствие числа и типа параметров в вызовах процедур и функций и т. д.

Допустим, что компиляция прервана, а на экран выведено сообщение об ошибке, как, например, показано на рис. 1.12.

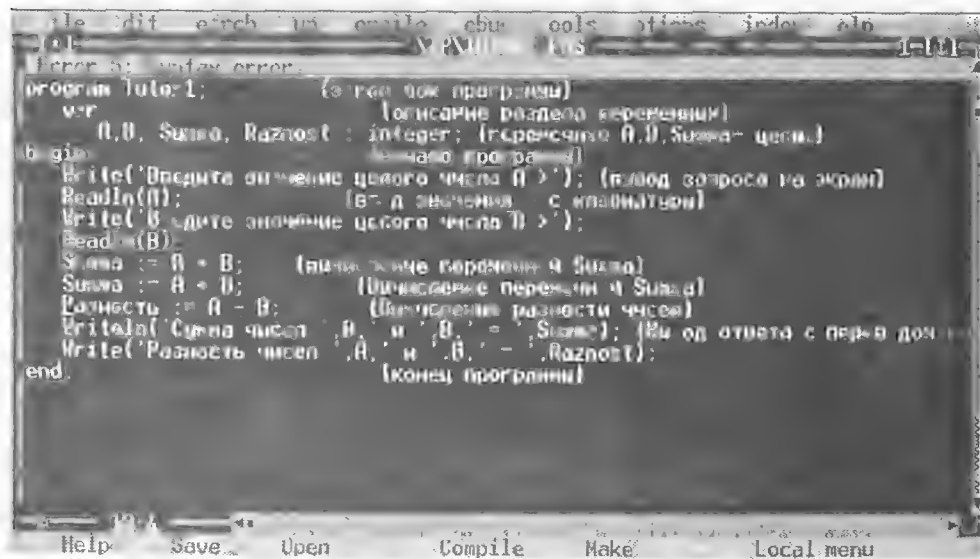


Рис. 1.12. Окно редактирования с выводом сообщения об ошибке компиляции

В данном случае сообщение об ошибке «Error 5: Syntax error» вызвано синтаксической ошибкой — записью имени переменной (идентификатора) «Разность» на русском языке. Место ошибки в тексте программы показано курсором.

ПРИМЕЧАНИЕ

Первое нажатие клавиши очистит это сообщение, а нажатие комбинации клавиш `Ctrl+Q W` будет показывать его снова до тех пор, пока вы не измените файл или не перекомпилируете его.

Для получения подсказки о данной ошибке следует нажать клавишу `F1`. После ознакомления со справочной информацией закройте окно подсказки клавишей `Esc`, исправьте ошибку и заново выполните компиляцию программы. Если в результате компиляции будет выведено сообщение об отсутствии ошибок в программе, то следует запустить ее на выполнение клавишами `Ctrl+F9`. В ответ на запрос компьютера введите значения целых чисел `A` и `B`, после чего будет выполнен расчет суммы и разности чисел `A` и `B`, а результаты выведены на экран. Чтобы посмотреть результат выполнения программы на экране пользователя, нажмите клавиши `Alt+F5`.

Для значений переменных `A = 12`, `B = 5` экран компьютера будет выглядеть, как показано на рис. 1.13.

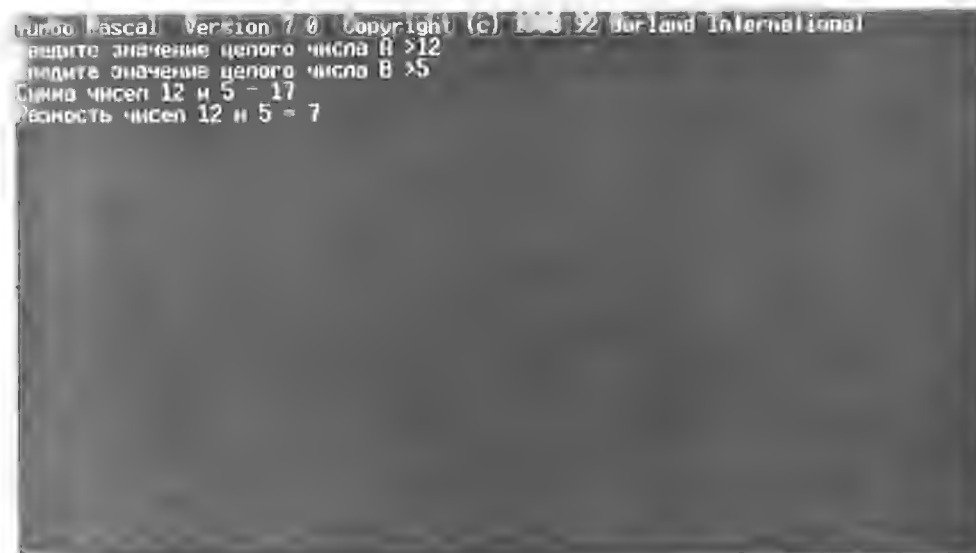


Рис. 1.13. Экран пользователя при выполнении программы из упражнения 2

Запишите программу на диск под именем `TUTOR2.PAS`, для чего клавишами `Alt+F` перейдите в меню `File` (см. табл. 1.2), выберите пункт `Save as...`, как показано на рис. 1.5. В окне ввода `Save file as` отредактируйте имя программы `TUTOR2`, нажимая клавиши `Tab` или `Shift+Tab`, перейдите к окну списка `Files` и установите в качестве текущего каталог `C:\TP\TUTOR`, в который требуется записать файл с текстом программы. После этого нажмите кнопку `OK`. Файл записан.

Контрольные вопросы и задания

Вопросы

1. Каковы возможности и в чем преимущества интегрированной среды программирования Turbo Pascal 7.0?
2. Перечислите основные файлы среды программирования Turbo Pascal и их назначение. Как запустить среду программирования Turbo Pascal?
3. Перечислите основные компоненты окна редактирования программ среды программирования Turbo Pascal. В чем их назначение?
4. Каково назначение пунктов File, Edit, Run, Compile главного меню среды программирования Turbo Pascal?
5. Каково назначение следующих опций пункта меню File: Open, Save As, DOS shell?
6. Каково назначение следующих опций пункта меню File: New, Save, Exit?
7. Опишите значение информации в строке состояния окна редактирования интегрированной среды программирования: F1 Help, F2 Save, F3 Open, Alt+F9 Compile, F9 Make, Alt+F10 Local menu.
8. Что такое локальное меню, какие локальные меню имеются в интегрированной среде программирования? Как их вызывать?
9. Как откомпилировать программу?
10. В чем отличие пункта Run от пункта Compile главного меню интегрированной среды программирования?
11. Как посмотреть результаты выполнения программы в окне пользователя?
12. Каково назначение информационно-справочной системы среды программирования Turbo Pascal? Почему ее называют контекстно-ориентированной? Как осуществляется управление системой помощи?

Задания

1. Пользуясь интегрированной средой программирования Turbo Pascal, считайте с диска программу TUTOR1.PAS, измените ее таким образом, чтобы она вместо суммы двух целых чисел вычисляла их произведение и записывала его значение в память под именем Proizved. Откомпилируйте программу, проверьте ее действие для значений переменных: A = 6, B = -3. Запишите измененную программу на диск под именем TUTOR3.PAS.
2. Измените программу из задания 1 таким образом, чтобы она вычисляла площадь прямоугольника по двум введенным сторонам и выводила на экран сообщение о результате вычислений.

Глава 2

Основные элементы языка Pascal

Алфавит и словарь языка Pascal

Любой язык представляет собой совокупность символов, соглашений и правил, используемых для общения. При записи алгоритма решения задачи на языке программирования необходимо четко знать правила написания и использования элементарных информационных и языковых единиц. Основой Pascal, как и любого языка, является алфавит — конечный набор знаков, состоящий из букв, десятичных и шестнадцатеричных цифр, а также специальных символов.

Символы в Pascal

В качестве букв в Pascal используются прописные и строчные буквы латинского алфавита:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

и знак подчеркивания (_);

в качестве десятичных цифр: 0 1 2 3 4 5 6 7 8 9.

Шестнадцатеричные цифры включают десятичные цифры и буквы от A до F (или от a до f).

При написании программ применяются следующие специальные символы:

+	Плюс	,	Запятая
-	Минус	.	Точка
*	Звездочка	:	Двоеточие
/	Дробная черта	[]	Квадратные скобки
>	Больше	{ }	Фигурные скобки
<	Меньше	\$	Знак денежной единицы
=	Равно	()	Круглые скобки
;	Точка с запятой	^	Тильда
#	Номер	@	Коммерческое а
	Апостроф	нет обозначения	Пробел

Комбинации специальных символов могут образовывать составные символы:

:=	Присваивание	<=	Меньше или равно
<>	Не равно	>=	Больше или равно
	Диапазон значений	(. .)	Альтернатива []
(* *)	Альтернатива { }		

В программе эти пары символов нельзя разделять пробелами, если они используются в качестве знаков операций отношения или ограничителей комментария.

ПРИМЕЧАНИЕ

Русские буквы в программе должны заключаться в апострофы, например 'Пример текста на русском языке'.

Примеры:

W — прописная латинская буква;

R — прописная латинская буква;

w — строчная латинская буква;

9 — цифра;

\$ — специальный символ;

<-> — составной символ.

Слова в Pascal

Некоторые последовательности знаков алфавита образуют слова, отделенные друг от друга разделителями и несущие определенный смысл в программе. Разделителями могут служить пробелы, символы конца строки или комментарии. Набор слов, используемый в Pascal, можно разделить на три группы: зарезервированные слова, стандартные идентификаторы и идентификаторы пользователя.

Зарезервированные слова являются составной частью языка, имеют фиксированное начертание и раз и навсегда определенный смысл. Они не могут изменяться программистом. Зарезервированные слова версии языка Pascal для персональных ЭВМ приведены ниже.

Зарезервированные слова версии языка Pascal

absolute	Абсолютный	label	метка
and	Логическое И	library	библиотека
array	Массив	mod	остаток от деления
asm	Ассемблер	nil	Отсутствие
begin	Начало блока	not	логическое НЕ

case	Вариант	or	Логическое ИЛИ
const	Константа	of	Из
constructor	Конструктор	object	Объект
div	Деление нацело	packed	Упакованный
go to	Переход на	procedure	Процедура
do	Выполнять	program	Программа
downto	Уменьшить до	record	Запись
destructor	Деструктор (разрушитель)	repeat	Повторять
else	Иначе	set	Множество
end	Конец блока	shl	Сдвиг разрядов влево
exports	Экспорт	shr	Сдвиг разрядов вправо
external	Внешний	string	Строка
file	Файл	then	То
for	Для	to	Увеличивая
forward	Опережающий	type	Тип
function	Функция	unit	Модуль
if	Если	until	До
implementation	Реализация	uses	Использовать
in	В (входит в ...)	var	Переменная
inline	Основной	while	Пока
interrupt	Прерывание	with	С
interface	Интерфейс	xor	Исключающее ИЛИ
inherited	Наследование		

ВНИМАНИЕ

Зарезервированные слова нельзя использовать в качестве имен, вводимых программистом для обозначения величин, и т. д.

Группа слов, имеющая определенный смысл, называется словосочетанием. В языке программирования словосочетание, состоящее из слов и символов и задающее правило вычисления некоторого значения, называется *выражением*. Минимальная конструкция языка, представляющая собой законченную мысль, есть *предложение*. Если предложение языка программирования задает полное описание некоторого действия, которое необходимо выполнить, оно называется *оператором*. Предложение, описывающее структуру и организацию данных — объектов языка, над которыми производятся различные действия, называется *описанием*.

Чтобы научиться правильно писать программы для компьютера, необходимо изучить синтаксис языка программирования (правила записи его конструкций) и его семантику (смысл и правила использования этих конструкций).

Формальные методы описания синтаксических конструкций языка программирования

Для описания синтаксических конструкций языка программирования в настоящее время наиболее распространены два формальных метода. Первый использует форму записи, предложенную Джоном Бэкусом и Питером Нау-ром, когда они описывали синтаксис языка Алгол-60. С тех пор эта форма называется Backus Naur Form, или сокращенно BNF.

Другой формальный метод, наглядно представляющий синтаксические конструкции языка в графическом виде, использует *синтаксические диаграммы*. Популяризировал синтаксические диаграммы создатель языка Pascal Н. Вирт, и поэтому их часто называют синтаксическими диаграммами Вирта.

На синтаксических диаграммах используются два вида четырехугольников — с прямыми и скругленными углами (иногда их заменяют кружками или овалами). В прямоугольники заключаются элементы языка, значение которых должно быть определено (так называемые *нетерминальные* символы). В четырехугольниках со скругленными углами (или кружках, овалах) размещаются так называемые *терминальные* (базовые) символы, или пероглифы языка, значение которых в определении не нуждается. Направление движения по диаграмме при раскрытии структуры понятия, записанного при входе в диаграмму, указывают стрелки.

Например:



Чтобы получить правильные грамматические конструкции языка, используя синтаксические диаграммы, нужно идти по путям, указанным стрелками, от одного четырехугольника к другому до тех пор, пока не встретится выход. Там, где предусмотрено более одного направления движения, можно выбирать любое. Если по пути встречается ссылка к другой синтаксической диаграмме, то следует войти в эту новую диаграмму, пройти по ней, выйти из нее и возвратиться на старое место в первоначальной диаграмме. Если по пути движения встречается точка, то это означает, что данный путь характерен только для Turbo Pascal и является расширением стандарта языка.

Варианты представления синтаксических конструкций языка программирования методом BNF или методом синтаксических диаграмм являются тождественными.

Идентификаторы

Чтобы программа решения задачи обладала свойством массовости, следует вместо конкретных значений величин использовать их обозначения, чтобы

иметь возможность изменять их значения по ходу выполнения программы. Для обозначения переменных и постоянных величин, различных процедур, функций, объектов используются имена — идентификаторы, устанавливающие соответствие между объектом и некоторым набором символов.

Стандартные идентификаторы

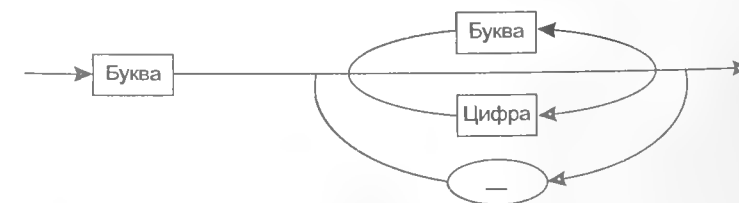
Для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций служат стандартные идентификаторы, например: integer, Sin, Cos, Ln, Sqr, Sqrt, Read, Readln, Write, Writeln. В этом примере стандартный идентификатор Sin вызывает функцию, вычисляющую синус заданного угла, Read, Readln вызывают процедуру, организующую ввод данных, Write, Writeln вызывают процедуру, организующую вывод данных. Любой из стандартных идентификаторов, в отличие от зарезервированных слов, можно переопределить, но это чаще всего приводит к ошибкам. Поэтому на практике стандартные идентификаторы лучше использовать без каких-либо изменений.

Пользовательские идентификаторы

Для обозначения меток, констант, переменных, процедур и функций, определенных самим программистом, применяются пользовательские идентификаторы. При этом идентификаторы в программе должны быть уникальными, т. е. в каждом блоке программы один идентификатор не может использоваться для обозначения более чем одной переменной или постоянной величины, и т. д.

Компилятор Turbo Pascal строго следит за этим, и если это требование не соблюдается, то компиляция прерывается, а на экран выводится сообщение об ошибке «Error 4: Duplicate identifier» и указывается дублирующийся идентификатор.

Синтаксическая диаграмма понятия «идентификатор» выглядит следующим образом:



ВНИМАНИЕ

В идентификатор не могут входить пробелы и специальные символы. Обратите внимание, что буквы русского алфавита не могут входить в идентификатор Turbo Pascal.

При написании программ следует соблюдать общие правила написания идентификаторов.

1. Идентификатор может начинаться только с буквы или знака подчеркивания (исключение составляют метки, которые могут начинаться также и с цифры).
2. Идентификатор может состоять из букв, цифр и знаков подчеркивания (пробелы, точки и другие специальные символы недопустимы).
3. Между двумя идентификаторами должен быть по крайней мере один пробел.
4. Максимальная длина идентификатора составляет 127 символов, но значимыми являются только первые 63 символа.
5. При записи идентификаторов можно использовать как прописные, так и строчные буквы. Компилятор не делает различий между ними, хотя они и имеют различные ASCII-коды. На практике рекомендуется применять эту особенность для более простого чтения и понимания значений идентификаторов. Так, вместо идентификатора `nomerotdela` лучше написать `NomerOtdela`, выделив прописными буквами каждую из двух смысловых частей.

Правильно выбранные идентификаторы значительно облегчают чтение и понимание программы, а также уменьшают вероятность появления ошибок при модификации программ. Например, значение даты удобнее обозначить идентификатором `Data`, чем просто буквой `D` или любым другим символом.

Примеры:

`Metka12`

`2graph` — ошибка, идентификатор начинается с цифры

`Block_56`

`Nomer.Doma` — ошибка, идентификатор содержит точку

`Сумма` — ошибка, идентификатор содержит буквы русского алфавита.

Константы и переменные

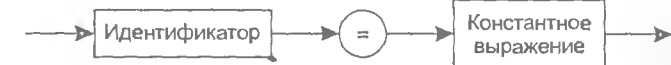
Как и в других языках программирования, в Pascal данные делятся на константы и переменные. В программе константы и переменные определяются идентификаторами (именами), по которым к ним можно обращаться для получения текущих значений.

Константами называются элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы не изменяются. Константы задаются пользовательскими идентификаторами. Например, если вы используете в программе ваше имя, то его лучше всего задать константой, так как имя не меняет своего значения.

Синтаксическая диаграмма определения константы выглядит следующим образом:



Определение константы:



Все константы должны быть описаны в специальном разделе, который начинается зарезервированным словом `const` (`constant` — константа).

Формат:

`const`

`<идентификатор> = <значение константы>;`

Например:

`const`

`MyName = 'Петя Иванов';`

`MyBirthDay = '27 августа 1950 г';`

`Max = 1000;`

`Min = 0;`

`Center = ( Max - Min ) / 2;`

`Num_School = 86;`

В Pascal имеется ряд констант, к значениям которых можно обращаться без предварительного определения. Их называют *зарезервированными константами*. Наиболее употребительные из них приведены в табл. 2.1.

Таблица. 2.1. Зарезервированные константы

Идентификатор	Тип	Значение	Описание
<code>True</code>	boolean	<code>True</code>	"Истина"
<code>False</code>	boolean	<code>False</code>	"Ложь"
<code>Maxint</code>	integer	32767	Максимальное целое

Переменными называют величины, которые могут менять свои значения в процессе выполнения программы. Каждая переменная или константа принадлежит к определенному типу данных. Тип констант автоматически распознается компилятором без предварительного описания.

Тип переменных должен быть описан перед тем, как с переменными будут выполняться какие-либо действия. Тем самым мы как бы сообщаем компьютеру, какие ячейки памяти следует использовать для хранения данных в программе.

Само название «переменная» подразумевает, что содержимое объявленной области памяти будет изменяться в ходе выполнения программы. Переменные описываются в специальном разделе, который начинается зарезервированным словом `var` (*variable* — переменная). Формат:

```
var
  <идентификатор> : <тип>;
```

Пример.

```
var
  A, B : integer;
  Summa : real;
```

Имя переменной подобно ящичку, который можно заполнить различными значениями, чего нельзя сделать с константой.

Синтаксическая диаграмма определения переменных выглядит следующим образом:



Кроме констант и переменных существуют так называемые *типизированные константы*, которые являются своеобразным промежуточным звеном между переменными и константами. Слово «константа» означает, что данные этого типа описываются в разделе `const`, а слово «типизированная» указывает, что для них должен указываться и тип, как у переменных. Формат:

```
const
  <идентификатор> : <тип> = <значение>;
```

Пример:

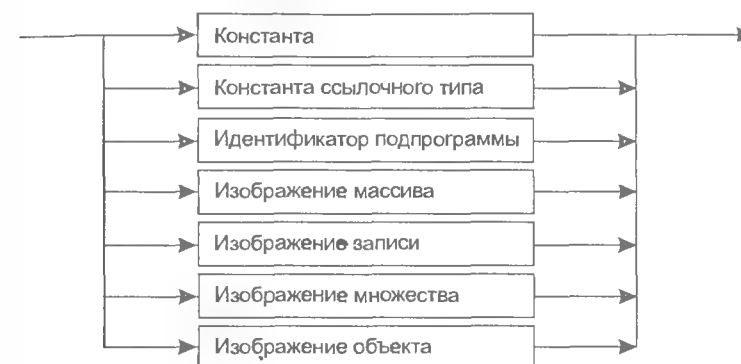
```
const
  VideoSeg : word = $B800;
  Ocenka : byte = 4;
  Predmet : string = 'Информатика';
```

Синтаксическая диаграмма определения типизированных констант записывается следующим образом:



В прикладном аспекте типизированная константа равнозначна переменной с заранее инициализированным значением, и в программе действия с ней могут производиться так же, как с переменной.

Типизированная константа:



СОВЕТ

Имена констант и переменных должны соответствовать их назначению, в этом случае программа будет проще для понимания.

Структура Pascal-программы

Программа реализует алгоритм решения задачи. В ней программист записывает последовательность действий, выполняемых над определенными данными с помощью определенных операций для реализации заданной цели. Основными характеристиками программы являются: точность полученного результата, время выполнения и объем требуемой памяти. О соответствии этих показателей решаемой задаче и возможностям компьютера должен позаботиться сам программист. В большинстве случаев определяющим требованием является точность. Ограничения по объему памяти и времени выполнения носят менее жесткий характер.

Программа на языке Pascal состоит из строк. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования Turbo Pascal или любого другого редактора. В первом случае программа может после выхода из редактора (при нажатии клавиши F10) в главное меню компилироваться и выполняться; во втором случае программу следует записать в файл и затем вызвать для компиляции и выполнения в интегрированной среде программирования Turbo Pascal.

Набирая текст программы, программист может произвольно располагать строки на экране. Строка может начинаться с любой колонки, т. е. величина отступа от левой границы экрана для каждой строки устанавливается самим программистом с целью получить наиболее удобный для чтения текст программы. Количество операторов в строке произвольно, но если в строке записывается один оператор, то такая программа легче читается.

Существуют различные схемы написания программ на языке Pascal, все они отличаются количеством отступов слева в каждой строке и различным использованием прописных букв. В данном пособии применяется следующая схема:

- зарезервированные слова `program`, `procedure`, `function` пишутся строчными буквами;
- имена констант, переменных, процедур, функций начинаются с прописных букв;
- операторы записываются только строчными буквами;
- логически подчиненные структуры записываются на одну строку ниже и на одну или две позиции правее по отношению к более старшим.

Такая схема записи создает условия для лучшего понимания программы и значительно более быстрого обнаружения в ее тексте ошибок.

Следует учитывать, что максимальный размер программы на Pascal ограничен. Компилятор позволяет обрабатывать программы и библиотечные модули, в которых объем данных и генерируемый машинный код не превышают 64 Кбайт каждый. Если программа требует большего количества памяти, следует использовать библиотечные модули (.TPU-файлы) или оверлейные структуры.

ПРИМЕЧАНИЕ

Оверлеи — части исполняемой программы, которые используют одну и ту же область оперативной памяти. В каждый момент времени в памяти может находиться только один оверлей, в зависимости от выполняемой функции. В процессе выполнения программы эти части могут замещать друг друга в памяти.

Синтаксически программа состоит из необязательного заголовка и блока.

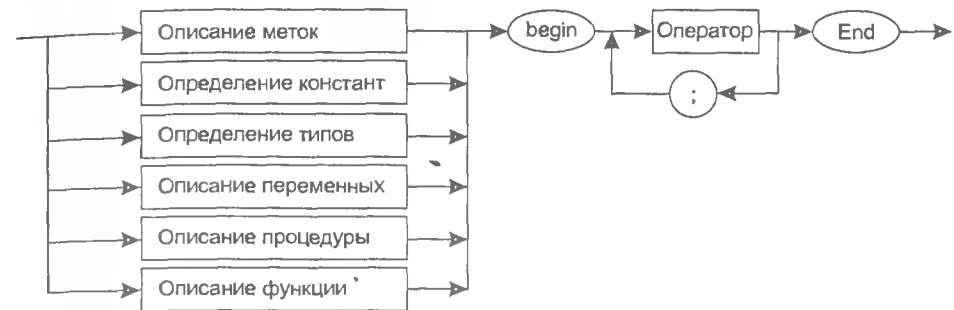


Блок может содержать в себе другие блоки. Блок состоит из двух частей: описательной и исполнительной. Первая часть может отсутствовать, без второй блок не имеет смысла. Блок, который не входит ни в какой другой блок, называется *глобальным*. Если глобальный блок содержит другие блоки, то они называются *локальными*. Глобальный блок — это основная программа, он должен присутствовать в любом случае. Локальные блоки — это процедуры и функции, их присутствие необязательно. Объекты программы (типы, переменные, константы и т. д.) тоже называются глобальными и локальными. Областью действия объектов является блок, в котором они описаны, и все вложенные в него блоки.

Блочная структура обеспечивает структуризацию программ на уровне исходных текстов. В идеальном случае программа на языке Pascal состоит из про-

цедур и функций, которые вызываются для выполнения из раздела операторов основной программы.

Синтаксическая диаграмма блока выглядит следующим образом:



Исходя из этого можно записать структуру программы следующим образом:

```

program <имя> (Input,Output);
  uses <имя1, имя2...>;
  label
  const .. :
  type
  var ....;
  procedure <имя>;
    <тело процедуры>
  function <имя>;
    <тело функции>
begin
  . <операторы>
end.
  
```

Рассмотрим структуру программы на примере программы решения задачи вычисления произведения двух целых чисел, созданной при выполнении упражнения во второй главе.

```

program Tutor3;                                {Заголовок программы}
var                                              {Описание раздела переменных}
  A,B, Proizved : integer; {Переменные A,B,Proizved – целые}
begin                                           {Начало программы}
  write('Введите значение целого числа A >'); {Вывод запроса на экран}
  Readln(A);                                  {Ввод значения A с клавиатуры}
  Write('Введите значение целого числа B >');
  Readln(B);
  Proizved := A * B;                          {Вычисление переменной Proizved}
  Write('Произведение чисел '.A.' и '.B.' = '.Proizved); {Вывод ответа}
end.                                           {Конец программы}
  
```

В начале программы находится заголовок, состоящий в общем случае из зарезервированного слова `program`, имени программы Tutor3 и параметров, с помощью которых программа взаимодействует с операционной системой.

Заголовок программы не является обязательным и может отсутствовать, однако рекомендуется всегда его записывать для быстрого распознавания нужной программы среди листингов других программ. Параметрами программы обычно являются идентификаторы стандартных файлов ввода-вывода Input и Output (в программах на Turbo Pascal их можно не указывать).

После заголовка следует программный блок, состоящий в общем случае из семи разделов:

- списка имен подключаемых библиотечных модулей (он определяется зарезервированным словом `uses`);
- описания меток;
- описания констант;
- определения типов данных;
- описания переменных;
- описания процедур и функций;
- операторов.

Любой раздел, кроме раздела операторов, может отсутствовать. Разделы описаний (кроме `uses`, который всегда располагается после заголовка программы) могут встречаться в программе любое количество раз и следовать в произвольном порядке. Главное, чтобы все описания объектов программы были сделаны до того, как эти объекты будут использованы.

Раздел `uses`

Этот раздел состоит из зарезервированного слова `uses` и списка имен подключаемых стандартных и пользовательских библиотечных модулей.

Формат:

```
uses <имя1>,<имя2>,...;
```

Пример:

```
uses Crt, Dos, MyLib;
```

Подробно о структуре и организации библиотечных модулей будет рассказано ниже.

Раздел описания меток

Перед любым оператором языка Pascal можно поставить метку, что позволяет выполнить прямой переход на этот оператор с помощью оператора перехода `go to` из любого места программы.

ВНИМАНИЕ

Нельзя выполнять переход на оператор, находящийся в теле цикла, а также внутри составного оператора.

Метка состоит из имени и следующего за ним двоеточия. Именем может служить идентификатор или цифра. Максимальная длина имени метки ограничена 127 символами. Перед использованием метка должна быть описана. Раздел описания меток начинается зарезервированным словом `label` (метка), за которым следуют имена меток, разделенные запятыми. За последним именем ставится точка с запятой.

Формат:

```
label <им>
```

Пример:

```
label
  Metka1, Metka2, l11, Blok10;
```

После записи метки в разделе операторов следует двоеточие, показывающее компилятору, что идентификатор используется в качестве метки:

```
label
  M1, M2:      {Описание меток}
begin

  M1: <оператор> {Использование M1 в разделе операторов}

  M2: <оператор> {Использование M2 в разделе операторов}
end.
```

Если метка описана, но не используется в разделе операторов, то ошибки при этом не возникает, т. е. метки можно описывать и применять по мере расширения программы.

Раздел описания констант

В разделе описания констант производится присваивание идентификаторам констант постоянных значений. Раздел начинается зарезервированным словом `const`, за которым следует ряд выражений, присваивающих идентификаторам постоянные числовые или строковые значения. Выражения присваивания отделяются друг от друга точкой с запятой. Формат:

```
const <идентификатор> = <значение>;
```

Пример:

```
const
  MaxInd: word = 100; {Типизированная константа}
  Name = 'Петя';      {Строковая константа}
  Code = $124;         {Константа – шестнадцатеричное значение}
```

ПРИМЕЧАНИЕ

Удачное с точки зрения мнемоники именованное констант делает программу лучше читаемой и позволяет быстро вносить в нее изменения при изменении алгоритма

В Turbo Pascal имеется большое число стандартных констант, к которым можно обращаться без предварительного описания.

Раздел описания типов данных

Тип данных может быть либо описан непосредственно в разделе описания переменных, либо определяться идентификатором типа. Стандартные типы не требуют описания, в отличие от типов, определенных пользователем. Строго говоря, синтаксис языка Pascal не требует обязательного определения идентификатора типа и в последнем случае, так как тип можно задать перечислением в разделе описания переменных. Выбор описания типа зависит, таким образом, только от программиста и специфики программы.

Раздел описания типов данных начинается зарезервированным словом `type`, за которым следуют одно или несколько определений типов, разделенных точкой с запятой. Формат:

```
type <имя типа> = <значения типа>;
```

Пример:

```
type
  LatLetter = ('A'..'Z');
  Days = 1..31;
  Matr = array[1..10] of integer;
```

Каждое описание задает множество значений и связывает с этим множеством некоторое имя типа. Например, в данном описании тип `LatLetter` определяет множество букв латинского алфавита, `Days` — множество целых чисел от 1 до 31, `Matr` — массив из 10 целых чисел.

Типы данных, разрешенные в языке Pascal, и их описание рассмотрены в главе 3.

Раздел описания переменных

Каждая встречающаяся в программе переменная должна быть описана. Описание обязательно должно предшествовать использованию переменной. Раздел описания переменных начинается зарезервированным словом `var` (variable — переменная), затем через запятую перечисляются имена переменных и после двоеточия следуют их тип и точка с запятой. Формат:

```
var
  <идентификатор,...> : <тип>;
```

В рассматриваемом примере программы три переменные `A`, `B` и `Proizved`, которые могут принимать целочисленные значения, описаны следующим образом:

```
var
  A,B, Proizved : integer;
```

Раздел описания процедур и функций

В этом разделе размещаются тела подпрограмм. *Подпрограммой* называется программная единица, имеющая имя, по которому она может быть вызвана из других частей программы. В языке Pascal роль подпрограмм выполняют процедуры и функции. В общем случае подпрограмма имеет ту же структуру, что и программа. Для описания подпрограмм используются зарезервированные слова `procedure` и `function`, которые записываются в начале подпрограммы.

Формат процедуры:

```
procedure <имя процедуры> {<параметры>};
  <разделы описаний>
  <раздел операторов>
end;
```

Формат функции:

```
function <имя функции> {<параметры>} : <тип результата>;
  <разделы описаний>
  <раздел операторов>
end;
```

Процедуры и функции подразделяются на стандартные и определенные пользователем. Стандартные процедуры и функции являются частью языка и могут вызываться без предварительного описания. Описание пользовательских процедур и функций обязательно. Более подробно процедуры и функции рассмотрены в главе 5 «Процедуры и функции».

Раздел операторов

В программе на языке Pascal раздел операторов является основным, так как именно в нем с предварительно описанными переменными, константами, значениями функций выполняются действия, позволяющие получить результат, ради которого создавалась программа.

Раздел операторов начинается зарезервированным словом `begin` (начало), далее следуют операторы языка, отделенные друг от друга точкой с запятой. Завершает раздел зарезервированное слово `end`. (конец) с точкой.

Например:

```
begin
  Write('Введите значение целого числа A >');
  Readln(A);
  Write('Введите значение целого числа B >');
  Readln(B);
  Proizved := A * B;
  Write('Произведение чисел ',A,' и ',B,' = ',Proizved);
end.
```

{Начало программы}
{Вывод запроса на экран}
{Ввод значения A с клавиатуры}
{Вычисление переменной Proizved}
{Вывод ответа}
{Конец программы}

Операторы выполняются строго последовательно, в том порядке, в котором они записаны в тексте программы в соответствии с синтаксисом и правилами пунктуации.

Слова `begin` и `end` являются аналогами открывающей и закрывающей скобок в обычных арифметических выражениях. Подробно операторы и правила их написания рассматриваются в главе 4 «Операторы».

Комментарии

Для лучшего понимания программы в нее включается пояснительный текст — комментарий. Комментарий можно записать в любом месте программы, где разрешено располагать пробелы. Текст комментария ограничивается символами `{ }` или `(*)` и может содержать любые комбинации латинских и русских букв, цифр и других символов алфавита языка Pascal. Ограничений на длину комментария нет, он может занимать несколько строк.

Примеры:

```
{ программа } или {начало программы*}
{вывод запроса на экран}
{Ввод значения A с клавиатуры}
{Вычисление произведения двух целых чисел Proizved}
{комментария занимающего
  ...}
```

В ограничителях `(*)` пробелы между скобкой и звездочкой недопустимы. В тексте комментария не должно быть знаков ограничителей, с которых начинается комментарий. Например, текст комментария {Пример {1} задания {4}} вызовет ошибку при компиляции. Однако ограничители `{ }` могут быть вложенными в `(*)`, и наоборот: `(*Пример{1}задания {4} *)` или `{Пример (* 1 *) задания (* 4 *)}`.

Комментарий не нормируется компилятором и поэтому не оказывает никакого влияния на программу. По месту расположения в программе комментарии можно разделить на четыре класса: объясняющие назначение программы, поясняющие смысл идентификаторов переменных и констант, описывающие логически обособленные части программы, и объясняющие сложные для понимания элементы алгоритма. В удачно прокомментированной программе легко найти ошибку, проанализировав несоответствие между замыслом автора (в комментариях) и реализацией (в тексте программы).

Ограничители `{ }` и `(*)` удобно использовать при отладке программ. В процессе отладки часто требуется временно исключить выполнение какой-либо части программы. Конечно, этого можно добиться, уничтожив временно ненужные операторы или обойдя их с помощью оператора `go to`. Однако оба этих способа непрактичны по ряду причин: повторный ввод вновь появившихся операторов, путаница с операторами `go to` и т. д. Гораздо удобнее

временно ограничить ненужную часть программы символами комментария `{ }` или `(*)`, тогда она будет восприниматься компилятором как комментарий и не будет исполняться.

Например:

```
{Начало программы}
writeln('Введите значение целого числа A >');
{Вывод запроса на экран}
writeln(A); {Ввод значения A с клавиатуры}
e( 'Введите значение целого числа B >');
writeln(' ');
{Вычисление переменной Proizved}
Proizved := A * B; {Вычисление переменной Proizved}
{Пример -- нег-полная часть программы}
('Произведение чисел 'A' и 'B' = 'Proizved);
writeln(' ');
{Конец программы}
```

При необходимости `{ }` или `(*)` можно убрать, и программа будет выполняться в полном объеме.

Директивы компилятора и управляющие символы

Текст программы может содержать директивы компилятора, которые используются для управления режимами компиляции.

ПРИМЕЧАНИЕ

Директива компилятора — компонент программы, управляющий последующей компиляцией программы.

Директивы, как и комментарии, заключаются в фигурные скобки, но они имеют отличительный признак — знак «\$», позволяющий компилятору интерпретировать их соответствующим образом.

Пример:

```
R-}
```

В рассматриваемой версии языка Pascal директивы компилятора имеют большое значение как на стадии отладки, так и при выполнении программы. По умолчанию директивы находятся в состоянии, гарантирующем минимальный объем объектного модуля и минимальные затраты времени на компиляцию.

В программе могут также встречаться управляющие символы «#» и «^». Знак «#» и следующий за ним целочисленное значение в диапазоне 0..255 обозначают символ таблицы ASCII, имеющий соответствующее десятичное значение.

Пример:

#23 - символ, имеющий десятичный код 23

#08 - символ, имеющий шестнадцатеричный код 08

Знак «^» и следующий за ним какой-либо другой символ трактуются компилятором как управляющий символ, т. е. «^» указывает, что далее следует один управляющий символ. Управляющие символы могут группироваться в строке без разделителей между ними. Допустимо использование управляющих символов вместе со строковыми данными.

Пример:

^I#25^Y^D — группа управляющих символов

Writeln('Обнаружена ошибка в тексте!',^G,^G);

Write(#234, #235, #236);

Библиотечные модули пользователя

Понятие библиотечного модуля является одним из основных в идеологии программных систем на языке Turbo Pascal. Именно они служат средством создания библиотек подпрограмм (процедур и функций). Библиотечный модуль — это результат компиляции в режиме Compile с установленной директивой Destination = Disk одной или нескольких процедур и функций. Модуль имеет имя, при упоминании которого в разделе uses любой программы можно получить доступ к каждой из находящихся в нем процедур или функций.

Создание библиотечного модуля требует определенной организации с применением зарезервированных слов unit, interface, implementation, begin, end. Система сама определяет структуру компилируемого файла и создает соответственно .TPU-файл (при обнаружении unit и т. д.) или .EXE-файл (при отсутствии unit, implementation и т. д.). В первом случае формируется библиотечный модуль, во втором — готовый к выполнению загрузочный модуль. Подробно структуру библиотечного модуля мы рассмотрим позднее.

Советы по стилю программирования

Для создания наглядных и легко читаемых программ на Pascal воспользуйтесь следующими советами.

1. Стандартизация стиля программирования заключается в том, что необходимо всегда придерживаться одного способа записи текста программы.
2. Для четкого указания вложенности управляющих структур требуется особым образом располагать операторы в тексте, так что служебные слова, которыми начинается и заканчивается тот или иной оператор, записываются на одном уровне, а все вложенные в него операторы записываются с отступом вправо. При записи конструкций языка более глубоких уровней вложенности следует сдвигать их от начала строки вправо. Каждое описание и каждый оператор следует писать с новой строки. Продолжение описаний и операторов на новые строки надо сдвигать вправо. Следует избегать длинных строк.

3. Рекомендуется любую программу сопровождать комментариями, поясняющими назначение всей программы и отдельных ее блоков, процедур, функций.
4. Имена для объектов программы надо выбирать так, чтобы они наилучшим образом соответствовали этим объектам, отражали их назначение.
5. Списки идентификаторов в блоках описания следует упорядочивать — это облегчает поиск в них нужных элементов.
6. Программирование сверху вниз. В процессе разработки алгоритма и программы следует начинать с самой общей модели решения, постепенно уточняя ее до уровня отдельного блока и затем детально прорабатывая каждый блок.

ПРИМЕЧАНИЕ

Иллюстрацией соблюдения данных правил могут послужить примеры программ в каталоге EXAMPLE, поставляемые фирмой Borland.

Упражнение 1. Загрузите интегрированную среду программирования, считайте с диска программу Tutor3. Измените ее таким образом, чтобы она выполняла целочисленное деление A на B и вычисляла остаток от деления A на B. Для этого в разделе описания переменных нужно ввести переменные Rezult и Ostatok целого типа:

var

A, B, Rezult, Ostatok : Integer; {Переменные A, B, Rezult, Ostatok — целые}

Операцию вычисления значений величин Rezult и Ostatok можно записать следующими операторами:

Rezult := A div B;

Ostatok := A mod B;

Вывод результатов работы программы на экран можно запрограммировать следующим образом:

Write('Результат целочисленного деления чисел '.A.' и '.B.' =', Rezult);
Write('Остаток от деления чисел '.A.' и '.B.' = '.Ostatok);

Откомпилируйте программу и проверьте ее работу, затем запишите отлаженный вариант на диск под именем Tutor4.

Контрольные вопросы и задания

1. Укажите буквы, символы, составные символы:
^ Y, <, +, *, R, k, \$, !, y
2. Что в списке можно рассматривать как идентификаторы:
F10, ФИО, 22222, X, Y, >=, &, \$, Summa, _Rezult

3. Укажите идентификаторы, которые проще воспринимаются при чтении, объясните причину:

```
klass1;
Klass_1;
summadoxoda;
SummaDoxoda;
nomerdoma;
Nomer_Doma.
```

4. Сколько в следующем списке зарезервированных слов:

X, Program, Y, Summa, MyMoney, Произведение, Vova, begin, end, if, repeat, Read?

5. В каких случаях следует использовать переменные:

- 1) если в программе используется какое-либо число;
- 2) если в вычислениях какой-либо операнд постоянно меняет свое значение;
- 3) если операнд в выражении хотя бы один раз меняет значение.

6. Какие заголовки программ правильны:

- 1) program Zarplata;
- 2) program Сумма;
- 3) program Summa Nalogov;
- 4) программа Teach_Kurs;
- 5) program 12Kurs2;
- 6) program Summa_Elementov?

7. Какая структура программы правильна:

```
1) program MyProgram;
   begin
       Writeln('Привет');
   end.
```

```
2) program MyFirst;
   begin
       X:=Y+100;
   end.
```

8. Какой из перечисленных разделов обязателен в программе:

- 1) раздел var?
- 2) раздел const?
- 3) раздел type?
- 4) раздел begin .. end.?
- 5) раздел label?

9. Какие из комментариев неправильны:

- 1) { Программа вычисляет логарифм введенного числа };
- 2) (* Это тоже комментарий *);

- 3) {{ Комментарий в комментарии }};
- 4) (* { И это комментарий в комментарии } *);
- 5) { (* Еще один вариант *) };
- 6) ((* Самый последний вариант *)*)).

10. Есть ли причины, по которым следующая программа не будет выполнена:

```
program Test;
begin
end.
```

11. Для чего используется слово uses?

- 1) такого слова нет в языке Turbo Pascal;
- 2) это пользовательский идентификатор;
- 3) с его помощью подключают стандартные библиотеки;
- 4) это стандартная константа, равная 3,14;
- 5) это логическая операция.

12. В разделе процедур и функций описываются только стандартные процедуры?

- 1) да, только стандартные;
- 2) нет, только пользовательские;
- 3) и стандартные, и пользовательские;
- 4) такого раздела вообще не может быть в программе.

13. Откомпилируйте простейшую программу:

```
program Tutor;
begin
    Writeln(5*6);
    Writeln('Привет');
end.
```

Каков будет результат ее выполнения?

Измените программу, чтобы результат равнялся 35.

14. Где ошибки (их три) в следующей программе?

```
program Ошибки1;
begin
    Summa:=6+8;
end.
```

Исправьте программу и добейтесь компиляции без ошибок

15. Где ошибки (их три) в следующей программе?

```
program Ошибки2;
{{ Программа с ошибками }}
begin
```

```
var X: integer;
X:=5 * 3;
Write(X);
end.
```

Исправьте программу и добейтесь компиляции без ошибок.

16. Где ошибка в следующей программе?

```
program MyError;
uses:
{(* Программа с ошибкой *)}
begin
end.
```

Исправьте программу и добейтесь компиляции без ошибок.

17. Напишите самую короткую программу.

18. Где ошибка в следующей программе?

```
program Kvadr;
begin
  WriteLn('Введите значение X');
  WriteLn('X в квадрате='.X*X);
end.
var X:integer;
```

Исправьте программу и добейтесь компиляции без ошибок.

19. Где ошибка в следующей программе?

```
program Summa;
var X,Y,Сумма:integer;
begin
  WriteLn('Введите значение X');
  ReadLn(X);
  WriteLn('Введите значение Y');
  ReadLn(Y);
  Сумма:=X+Y;
  WriteLn(Сумма);
end.
```

Исправьте программу и добейтесь компиляции без ошибок.

20. Создайте программу для вычисления площади прямоугольника по введенным в диалоге двум сторонам. Сохраните текст программы на диске под именем tutor5.pas, откомпилируйте и проверьте ее действие.

21. Создайте программу для вычисления длины окружности и площади круга по указанному радиусу.

22. Создайте программу для вычисления длин высот треугольника, у которого длины сторон равны A, B, C.

23. Создайте программу для вычисления длин медиан треугольника, у которого длины сторон равны A, B, C.

24. Создайте программу для вычисления величины работы, совершенной при равномерном подъеме груза массой M кг на высоту H м. Ускорение свободного падения опишите как константу $G = 9,81$.
25. Создайте программу для вычисления давления столба жидкости плотностью R и высотой H на дно сосуда.
26. Создайте программу для вычисления силы давления, действующей на пол со стороны стола массой M , если суммарная площадь опоры ножек стола равна 100 см^2 .
27. Создайте программу для вычисления выталкивающей силы, действующей на тело объемом V , наполовину погруженное в жидкость плотностью R .
28. Создайте программу для вычисления количества теплоты, полученного при сгорании M г керосина, если его теплота сгорания равна q .
29. Создайте программу для вычисления количества теплоты, затраченного на нагревание тела плотностью R , объемом V на T градусов Цельсия, если его удельная теплоемкость равна C .
30. Создайте программу для вычисления величины силы тока на участке электрической цепи сопротивлением R Ом при напряжении U В.
31. Создайте программу для вычисления напряжения на каждом из последовательно соединенных участков электрической цепи сопротивлением R_1, R_2, R_3 Ом, если сила тока при напряжении U В составляет I А.
32. Создайте программу для вычисления значения силы тока I на участке, состоящем из двух параллельно соединенных резисторов сопротивлением R_1 и R_2 , если напряжение на концах этого участка равно U .
33. Создайте программу, определяющую плотность тела, объем которого равен V , а масса — M .
34. Создайте программу, определяющую количество теплоты Q , требуемое для нагревания V л жидкости, взятой при температуре T_1 , до температуры кипения T_2 , если известна удельная теплоемкость жидкости q .

Глава 3

Типы данных. Ввод-вывод данных

Общие сведения

При решении задач выполняется обработка информации различного характера. Это могут быть целые и дробные величины, строки и др. Соответственно константы и переменные должны быть описаны как целые, дробные, строковые и т. д.

Для описания множества допустимых значений величины и совокупности операций, в которых может участвовать данная величина, используется указание ее типа данных. *Тип данных* (data type) — это множество величин, объединенных определенной совокупностью допустимых операций.

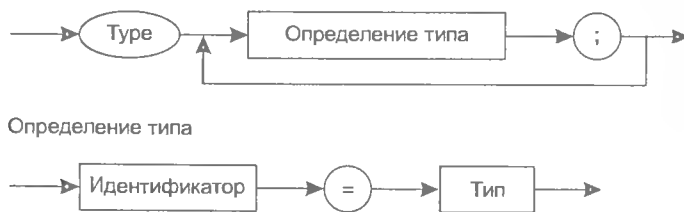
Каждый тип данных имеет свой диапазон значений и специальное зарезервированное слово для его описания. Например, значения 1 и 2 относятся к целочисленному типу, их можно складывать, умножать и выполнять над ними другие арифметические операции.

В языке Pascal для описания типа в общем случае используется зарезервированное слово `type`. Формат:

`type`

`<Имя типа> = <значения типа>;`

Синтаксическая диаграмма описания типов может быть представлена следующим образом:



Все типы данных можно разделить на две группы: скалярные и структурированные (составные). Скалярные типы, в свою очередь, делятся на стандартные и пользовательские.

Стандартные типы данных предлагаются пользователям разработчиками системы Turbo Pascal. К ним относятся целочисленные, вещественные, логические, булевские типы данных и указатели.

Пользовательские типы данных разрабатываются пользователями системы программирования Turbo Pascal.

Перечень типов данных в Turbo Pascal

Перечень типов данных в языке Turbo Pascal можно представить в виде следующей схемы:

1. Простые типы (скалярные типы).

Порядковые типы.

Целые типы:

`byte`,

`shortint`,

`integer`,

`word`,

`longint`.

Логический тип `boolean`.

Символьный тип `char`.

Перечисляемый тип.

Интервальный тип (диапазон)

Вещественные типы:

`real`,

`single`,

`double`,

`extended`,

`comp`.

Ссылочный тип.

2. Структурированные типы.

Строковый (`string`).

Регулярный (`array`).

Комбинированный (`record`).

Множественный (`set`).

Файловый (`file`).

3. Процедурные типы.

ПРИМЕЧАНИЕ

В Delphi список типов данных отличается от вышеприведенного, о чем будет рассказано во второй части книги.

Данные целочисленных типов могут быть представлены как в десятичной, так и в шестнадцатеричной системе счисления. Если число представлено в шестнадцатеричной системе, перед ним без пробела записывается знак «\$». Диапазон допустимых значений шестнадцатеричных чисел составляет от \$0000 до \$FFFF.

В десятичной системе числа могут записываться двумя способами: с фиксированной и с плавающей точкой.

Вещественные десятичные числа с фиксированной точкой записываются по обычным правилам арифметики. Целая часть от дробной отделяется десятичной точкой. Если десятичная точка отсутствует, число считается целым. Перед числом может находиться знак «+» или «-». Если знак отсутствует, то по умолчанию число считается положительным.

Примеры:

125 — целое десятичное число

\$1FF — шестнадцатеричное число

124.674 — вещественное число

-12.9 — отрицательное вещественное число

Вещественные десятичные числа в форме с плавающей точкой записываются в экспоненциальном виде: $mE+p$, где m — мантисса (целое или дробное число с десятичной точкой) E означает десять в степени, p — порядок (целое число).

Пример:

$5.18E+02 = 5.18 * 10^2 = 518$

$10E-03 = 10 * 10^{-3} = 0.01$

Пользовательские типы — перечисляемый и интервальный — разрабатываются самим программистом.

Структурированные типы в своей основе имеют один или несколько скалярных типов данных. К структурированным типам относятся строки, массивы, множества, записи, файлы и данные совершенно иной природы: процедурного типа и типа object. Двум последним типам трудно поставить в соответствие данные в обычном понимании этого слова. Разработчики из фирмы Borland назвали их процедурными типами и объектами, что точно соответствует их базовым признакам. Понимание работы с этими типами требует наличия определенного опыта и навыков программирования.

Скалярные типы данных

К скалярным (scalar — простые) типам данных относят типы данных таких величин, значения которых не содержат составных частей.

Целочисленные типы данных

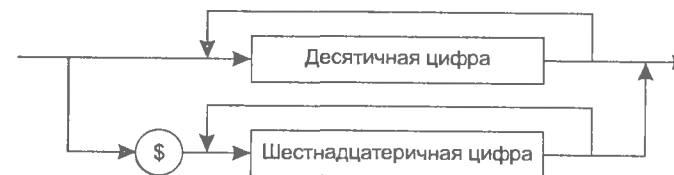
Целочисленные типы данных представляют собой значения, которые могут использоваться в арифметических выражениях и занимать в памяти от 1 до 4 байт (табл. 3.1).

Таблица 3.1. Целочисленные типы данных

Тип	Диапазон	Требуемая память (байт)
byte	0..255	1
shortint	-128..127	1
integer	-32768..32767	2
word	0..65535	2
longint	-2147483648..2147483647	4

Значения целых типов могут изображаться в программе двумя способами: в десятичном виде (традиционно, в виде последовательности цифр) и в шестнадцатеричном виде (в этом случае число предваряется знаком «\$», а цифры старше 9 обозначаются латинскими буквами от A до F).

Синтаксическая диаграмма для целых чисел выглядит следующим образом:



Пример:

```
var
  X1, X2: byte;
  Y1: word;
```

Над данными целого типа определены следующие арифметические операции: +, -, *, /, div, mod. Результат выполнения этих операций над целыми операндами также имеет целый тип.

Над данными целого типа определены следующие операции отношения: =, <>, <, >, <=, >=, вырабатывающие результат логического типа.

Для целых чисел определены следующие стандартные функции:

odd(x) — возвращает результат логического типа:

— для четного аргумента — false;

для нечетного — true;

succ(x) — возвращает следующее целое число (x + 1);

pred(x) — возвращает предыдущее целое число (x - 1);

ord(x) — возвращает аргумент x;

$\text{abs}(x)$ — возвращает модуль x ;
 $\text{chr}(x)$ — возвращает символ, ASCII-код которого равен x ;
 $\text{sqr}(x)$ — возвращает квадрат числа x ;
 $\text{sqrt}(x)$ — возвращает значение корня квадратного из x ;
 $\text{exp}(x)$ — возвращает e в степени x (экспоненту), результат вещественного типа;
 $\text{sin}(x)$ — возвращает синус x , результат вещественного типа;
 $\text{cos}(x)$ — возвращает косинус x , результат вещественного типа;
 $\text{ln}(x)$ — возвращает натуральный логарифм x , результат вещественного типа;
 $\text{arctan}(x)$ — возвращает арктангенс x , результат вещественного типа.

Для целых чисел определены следующие стандартные процедуры:

$\text{dec}(x, i)$ — уменьшает значение x на i , если i не задано, то на 1;
 $\text{inc}(x, i)$ — увеличивает значение x на i , если i не задано, то на 1.

Вещественные типы данных

Вещественные типы данных представляют собой вещественные значения, которые используются в арифметических выражениях и занимают в памяти от 4 до 6 байт. Pascal допускает представление вещественных значений и с плавающей, и с фиксированной точкой (табл. 3.2).

Таблица 3.2. Вещественные типы данных

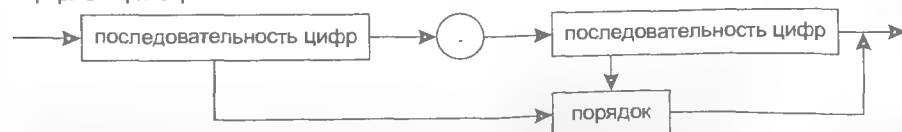
Тип	Диапазон	Мантисса	Требуемая память (байт)
real	$2.9 \cdot 10^E - 39 \dots 1.7 \cdot 10^{E38}$	11 – 12	6
single	$1.5 \cdot 10^E - 45 \dots 3.4 \cdot 10^{E38}$	7 – 8	4
double	$5.0 \cdot 10^E - 324 \dots 1.7 \cdot 10^{E308}$	15 – 16	8
extended	$1.9 \cdot 10^E - 4951 \dots 1.1 \cdot 10^{E4932}$	19 – 20	10
comp	$-2E+63 + 1 \dots 2E+63 - 1$	10 – 20	8

Вещественные значения могут изображаться в форме с фиксированной точкой, например 7.32, 456.721 или 0.015, а также в форме с плавающей точкой, т. е. парой чисел вида <мантисса>E<порядок>.

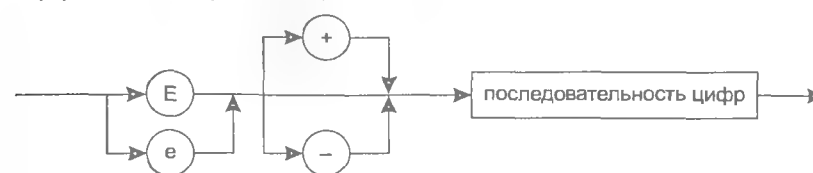
Числа из предыдущего примера в форме с плавающей точкой будут записаны так: 7.32E+00, 4.56721E+02, 1.5E-02.

Синтаксические диаграммы для записи вещественного числа будут выглядеть следующим образом:

в форме с фиксированной точкой:



в форме с плавающей запятой:



Над данными вещественного типа определены следующие арифметические операции: +, -, *, /. Результат выполнения этих операций также имеет вещественный тип.

Над данными вещественного типа определены следующие операции отношения: =, <>, <, >, <=, >=, вырабатывающие результат логического типа.

Для вещественных чисел определены следующие стандартные функции:

$\text{abs}(x)$ — возвращает модуль x , результат вещественного типа;
 $\text{chr}(x)$ — возвращает символ, ASCII-код которого равен x ;
 $\text{sqr}(x)$ — возвращает квадрат числа x , результат вещественного типа;
 $\text{sqrt}(x)$ — возвращает значение корня квадратного из x , результат вещественного типа;
 $\text{exp}(x)$ — возвращает e в степени x (экспоненту), результат вещественного типа;
 $\text{sin}(x)$ — возвращает синус x , результат вещественного типа;
 $\text{cos}(x)$ — возвращает косинус x , результат вещественного типа;
 $\text{ln}(x)$ — возвращает натуральный логарифм x , результат вещественного типа;
 $\text{arctan}(x)$ — возвращает арктангенс x , результат вещественного типа;
 $\text{trunc}(x)$ — преобразует вещественный аргумент x в целое число путем отбрасывания дробной части;
 $\text{round}(x)$ — преобразует вещественный аргумент x в целое число путем округления до ближайшего целого.

Выражение, составленное из переменных целого и вещественного типа, имеет вещественный тип. Допускается присваивание переменной вещественного типа значения выражения целого типа, но не наоборот.

ПРИМЕЧАНИЕ

Эффективное использование типов single, double, extended, comp возможно только при включенной директиве {\$N+}. По умолчанию она находится в выключенном состоянии {\$N-}.

Пример:

```
var
  Summa: single;
  Root1, Root2: double;
```

Упражнение 1. Измените программу Tutor3 таким образом, чтобы в результате ее выполнения вычислялось и выводилось на экран значение частного

двух целых чисел. Так как результат деления будет иметь вещественный тип, в раздел описания переменных данной программы должно быть добавлено его описание. Например, если идентификатор результата деления обозначить как `Ratio`, то раздел описания переменных в программе будет выглядеть следующим образом:

```
var
  A,B : Integer;
  Ratio : real;
```

Определение частного чисел `A` и `B` запишется операцией вещественного деления:

```
Ratio := A / B;
```

а вывод на экран результата можно задать следующим образом:

```
Writeln('Частное двух чисел равно ',Ratio);
```

Загрузите интегрированную среду программирования, считайте программу `Tutor3` с диска, отредактируйте ее и проверьте ее работу. При проверке работы программы попробуйте задать переменным следующие значения: `A=33000`, `B=33`.

Обратите внимание на то, что в результате вычислений получается не 1000, как вы ожидали, а `-9.8593939394E+02`, т. е. программа неправильно вычисляет результат арифметической операции. Причина ошибки в том, что мы, указав в разделе описания переменных для величин `A` и `B` тип `integer`, зарезервировали в памяти место только для хранения целого числа, принимающего значения из интервала `[-32768..32767]`, а задали для величины `A` значение 33000.

Внесите изменения в программу, указав в разделе описания для величин `A`, `B` тип `word` или `longint`, проверьте работу программы на примере больших значений чисел `A` и `B`. Сохраните программу на диске под именем `Tutor4`.

Этот пример наглядно демонстрирует необходимость правильного описания типов величин, обрабатываемых в программе.

Литерный (символьный) тип

Литерный (символьный) тип `char` определяется множеством значений кодовой таблицы компьютера. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII. Таблица кодов ASCII приведена в приложении А.

Для размещения в памяти переменной литерного типа требуется один байт.

Пример:

```
var
  Ch: char;
  Letter, Symbol: char;
```

В программе значения переменных и констант типа `char` должны быть заключены в апострофы. Например, `'A'` обозначает букву `A`, `' '` — пробел, `'.'` — точку с запятой.

Над данными символьного типа определены следующие операции отношения: `=`, `<>`, `<`, `>`, `<=`, `=>`, вырабатывающие результат логического типа.

Для данных символьного типа определены следующие стандартные функции: `chr(x)` — преобразует выражение `x` типа `byte` в символ и возвращает значение символа;

`ord(ch)` — преобразует символ `ch` в его код типа `byte` и возвращает значение кода;

`pred(ch)` — возвращает предыдущий символ;

`succ(ch)` — возвращает следующий символ.

Примеры:

```
ord(' ') = 58
ord('A') = 65
chr(128) = Б
pred('Б') = А
succ('Г') = Д
```

Булевский тип

Булевым типом называют тип данных, представляемый двумя значениями: `True` (истина) и `False` (ложь). Он широко применяется в логических выражениях и выражениях отношения. При описании величин этого типа указывают слово `boolean`. Для размещения в памяти переменной булевого типа требуется 1 байт.

Пример:

```
var
  Flag, Result: boolean;
```

Пользовательские типы

Кроме стандартных типов данных Pascal поддерживает скалярные типы, определенные самим пользователем. К ним относятся перечисляемый и интервальный типы.

Данные этих типов занимают в памяти один байт, поэтому скалярные пользовательские типы не могут содержать более 256 элементов. Их применение обеспечивает семантический контроль вводимых данных, значительно улучшает наглядность программы, делает более легким поиск ошибок и экономит память.

Перечисляемый тип

Перечисляемый тип (enumerated type) — тип данных, заданный списком принадлежащих ему значений.

Объявление перечисляемого типа описывает множество идентификаторов, которые являются возможными значениями перечисляемого типа. Идентификаторы в описании типа представляют собой константы. Отдельные значения указываются через запятую, а весь список заключается в круглые скобки. Первая константа имеет порядковый номер ноль, вторая — 1 и т. д.

Формат:

```
type
  <имя типа> = (<значение1, значение2..., значениеп>);
var
  <идентификатор...> : <имя типа>;
```

Синтаксическая диаграмма для перечисляемых типов имеет следующий вид:



Пример:

```
type
  Gaz = (Ge, C, O, N);
  Metall = (Na, K, Li, Cu, Zn);
var
  G1, G2, G3 : Gaz;
  Met1, Met2 : Metall;
  Season : (Winter, Spring, Summer, Autumn);
```

В данном примере приведены два явно описанных пользовательских типа данных — **Gaz** и **Metall**. Определены их значения — обозначения некоторых газов и металлов периодической таблицы Д. И. Менделеева. Переменные **G1**, **G2**, **G3** и **Met1**, **Met2** могут принимать только одно из перечисленных значений. Попытка присвоить им любое другое значение вызовет программное прерывание. Третий тип перечисления является анонимным (не имеет имени) и задается перечислением значений в разделе **var**. **Season** является переменной этого типа и может принимать значения **Winter**, **Spring**, **Summer** и **Autumn**. Таким образом может быть задан любой тип, но это не всегда приемлемо, так как первый способ более понятен и больше соответствует характеру языка Pascal. При этом имена внутри круглых скобок являются константами соответствующего типа перечисления и подчиняются обычным правилам для констант. Выражения и константы перечисляемого типа допустимы для использования в операторе **CASE**. Операции отношения и логические операции допустимы для значений перечисления одного и того же типа. Упорядочение осуществляется по номеру элемента в описании типа. Например, выражение **Winter < Spring** будет истинно, так как **Spring** имеет больший номер по порядку в описании типа, чем **Winter**.

ВНИМАНИЕ

В отличие от данных других типов Pascal не поддерживает операции ввода-вывода значений пользовательского перечисляемого типа. При необходимости программист сам должен организовать ввод-вывод таких данных.

Для работы с данными перечисляемого типа в языке Pascal предназначены стандартные подпрограммы **Succ**, **Pred**, **Ord**.

Описанные ранее переменные булевского типа можно представить и как перечисляемый тип, объявленный следующим образом:

```
type
  Boolean = (False, True);
```

Поэтому для значений **False** и **True** справедливы результаты вычисления выражений:

False < True	Succ(False) = True
Ord(False) = 0	Pred(True) = False
Ord(True) = 1	

Интервальный тип (диапазон)

Интервальный тип позволяет задавать две константы, определяющие границы диапазона значений для данной переменной. Компилятор при каждой операции с переменной интервального типа генерирует подпрограммы проверки, определяющие, остается ли значение переменной внутри установленного для нее диапазона.

Обе константы должны принадлежать одному из стандартных типов (напомним, что тип **real** здесь недопустим). Значение первой константы должно быть обязательно меньше значения второй.

Формат:

```
type
  <имя типа> = <константа1> .. <константа2>;
var
  <идентификатор...> : <имя типа>;
```

Синтаксическая диаграмма для интервальных типов выглядит следующим образом:



Пример:

```
type
  Days = 1 .. 31;
var
  RabDay, BolnDay : Days;
```

В этом примере переменные RabDay и BolnDay имеют тип Days и могут принимать любые значения из диапазона 1..31. Выход за границы диапазона вызывает программное прерывание.

СОВЕТ

Рациональнее определить интервальный тип более универсальным способом, задав границы диапазона не значениями констант, а их именами:

```
const
  Min = 1; Max = 31;
type
  Days = Min .. Max;
var
  RabDay, BolnDay : Days;
```

Структурированные типы данных

Структурированные типы данных определяют упорядоченную совокупность скалярных переменных и характеризуются типом своих компонентов. В языке Pascal допускаются следующие структурированные типы данных: строки, массивы, множества, записи, файлы, указатели, процедурные типы и объекты. Все они требуют отдельного рассмотрения и подробно изложены далее в книге.

Тождественность и совместимость типов

Для того чтобы в результате выполнения программы не получилось путаницы с величинами различного типа, например, когда цена корзины с продуктами равна расстоянию от дома до магазина, умноженному на количество этажей дома, программисту требуется знать и правильно применять понятия тождественности и совместимости типов величин, участвующих в операциях — операндов. Два типа являются тождественными, если они описаны вместе или если их определения используют один и тот же идентификатор типа.

Пример:

```
type
  M1, M2 = array[1..10] of byte; {M1, M2 — тождественные типы}
  S = set of byte;
  F = set of integer;           {S, F — нетождественные типы}
```

или

```
var
  A, B, Proizved : integer;
```

Тождественность типов требуется только для переменных фактических и формальных параметров при вызове процедур и функций. Совместимость типов играет важнейшую роль в выражениях и операциях сравнения и в операторах присваивания.

В операциях сравнения два типа являются совместимыми, если соблюдается хотя бы одно из следующих условий:

- оба типа являются одинаковыми;
- оба типа являются вещественными типами;
- оба типа являются целочисленными;
- один тип является поддиапазоном другого;
- оба типа являются поддиапазонами одного и того же основного типа;
- оба типа являются множественными типами с совместимыми базовыми типами;
- оба типа являются строковыми типами с одинаковым числом компонентов;
- один тип является строковым, а другой — строковым или символьным типом;
- один тип является указателем, а другой — любым типом указателей.

Пример:

'a' > 'b' {Допустимо, так как оба значения относятся к типу char};

'a' > 5 {Ошибка, так как сравниваемые значения имеют разные типы}.

В операциях присваивания два типа являются совместимыми, если соблюдается хотя бы одно из следующих условий:

- оба типа тождественны, и ни один из них не является файловым или структурным типом, содержащим компоненты с файловым типом на одном из своих уровней;
- оба типа являются совместимыми скалярными типами, и значения второго типа попадают в диапазон возможных значений первого;
- оба типа относятся к вещественным типам, и значения второго типа попадают в диапазон возможных значений первого;
- первый тип является вещественным, а второй — целочисленным;
- оба типа являются строковыми;
- первый тип является строковым, а второй — литерным;
- оба типа относятся к совместимым множественным типам, и все значения второго типа попадают в диапазон возможных значений первого типа;
- оба типа относятся к совместимым типам «указатель».

Пример:

```
var
  A, B: integer;
  C: real;

  A:= B: {Правильно}
  C:= B: {Правильно}
  A:= C: {Ошибка}
```

Выражения, операции, операнды

Конструкция языка, задающая порядок выполнения действий над элементами данных, называется *выражением*. Выражение состоит из операндов — величин и выражений, над которыми производится операция (константы и переменные всех типов, обращения к функциям), круглых скобок и знаков операций. Операции определяют действия, которые требуется выполнить над операндами. Например, в выражении $(X + Y - 10)$ X , Y и 10 — операнды; а «+», «-» — знаки операций сложения и вычитания.

В простейшем случае выражение может состоять из одной переменной или константы. Круглые скобки ставятся так же, как и в обычных арифметических выражениях для управления ассоциативностью и порядком выполнения операций.

Операции в языке Pascal делятся на арифметические, операции отношения, логические (булевские), операцию @, строковые и др. Выражения соответственно называются арифметическими, выражениями отношения, булевскими, строковыми и т. д. в зависимости от того, какого типа операнды и операции в них используются. Три первые группы операций описаны в данном подразделе, остальные будут рассмотрены при изучении соответствующих типов данных.

Тип значения, вычисляемого с помощью выражения, определяется типом его операндов и знаками выполняемых над ними операций.

Операции могут быть унарными и бинарными. В первом случае операция относится к одному операнду и всегда записывается перед ним, во втором — операция выражает отношение между двумя операндами и записывается между ними.

Например, $-A$ — унарная операция, $X + Y$ — бинарная.

Арифметические выражения и операции

Арифметическим называется выражение, составленное из операндов арифметического типа и использующее только знаки арифметических операций и круглые скобки. Порядок вычисления выражения определяется скобками и старшинством операций.

Арифметическое выражение порождает целое или действительное (вещественное) значение. Наиболее простыми формами арифметических выражений являются:

- целая или действительная константа без знака;
- целая или действительная переменная;
- элемент массива целого или действительного типа;
- функция, принимающая целое или действительное значение.

Значение переменной или элемента массива должно быть определено до их появления в арифметическом выражении. Другие арифметические выраже-

ния создаются на основе вышеперечисленных простых форм путем применения круглых скобок и арифметических операций.

Арифметические операции выполняют арифметические действия в выражениях над значениями операндов целочисленных и вещественных типов. Арифметические операции языка Pascal представлены в табл. 3.3.

Таблица 3.3. Арифметические операции

Операция	Действие	Типы операндов	Тип результата
Бинарные			
+	Сложение	Целый	Целый
		Вещественный	Вещественный
-	Вычитание	Целый	Целый
		Вещественный	Вещественный
*	Умножение	Целый	Целый
		Вещественный	Вещественный
/	Деление	Целый	Вещественный
		Вещественный	Вещественный
div	Целочисленное деление	Целый	Целый
mod	Остаток от деления	Целый	Целый
and	Арифметическое И	Целый	Целый
shl	Сдвиг влево	Целый	Целый
shr	Сдвиг вправо	Целый	Целый
or	Арифметическое ИЛИ	Целый	Целый
xor	Исключающее ИЛИ	Целый	Целый
Унарные			
+	Сохранение знака	Целый	Целый
-	Отрицание знака	Вещественный	Вещественный
		Целый	Целый
not	Арифметическое отрицание	Вещественный	Вещественный
		Целый	Целый

Операции сложения (+), вычитания (-), умножения (*) и деления (/) выполняются так же, как и в обычных арифметических выражениях.

Целочисленное деление (DIV) отличается от обычной операции деления тем, что вычисляет целую часть частного, а дробная часть отбрасывается. Перед выполнением операции оба операнда округляются до целых значений. Результат целочисленного деления всегда равен нулю, если делимое меньше делителя.

Например:

Выражение	Результат
11 DIV 5	2
10 DIV 3	3
2 DIV 3	0

Деление по модулю (MOD) вычисляет остаток, полученный при выполнении целочисленного деления. Например:

Выражение	Результат
10 MOD 5	0
11 MOD 5	1
10 MOD 3	1
14 MOD 5	4

Арифметическое И (AND) производит логическое умножение операндов в соответствии со следующей таблицей истинности:

1 AND 1 = 1	1 AND 0 = 0
0 AND 1 = 0	0 AND 0 = 0

Операнды записываются в десятичной форме, но во время выполнения переводятся в двоичную форму. Результат представляется в десятичной форме.

Упражнение 1. Вычислить результат выражения $A \text{ AND } B$, если $A = 12$ и $B = 22$. A и B занимают в памяти по два байта и в двоичной форме имеют вид: 0000000000001100 и 000000000010110. В результате выполнения операции 0000000000001100 AND 000000000010110 в соответствии с таблицей истинности получим результат 0000000000000100, или 4 в десятичной форме.

Следовательно, $12 \text{ AND } 22 = 4$.

Сдвиг влево (K SHL N) восстанавливает в качестве результата значение, полученное путем поразрядного сдвига на N позиций влево представленного в двоичной форме числа K.

Упражнение 2. Вычислить результат выполнения выражения $2 \text{ SHL } 7$. Число 2 занимает в памяти два байта и в двоичной форме имеет вид 0000000000000010. Сдвигаем каждый разряд на 7 позиций влево, получаем 0000000100000000, что соответствует числу 256 в десятичной форме.

Следовательно, $2 \text{ SHL } 7 = 256$.

Сдвиг вправо (SHR) выполняется аналогично с той лишь разницей, что сдвиг производится вправо. Например:

Выражение	Результат
160 SHR 2	40
90 SHR 2	22
256 SHR 7	2

Логическое сложение (OR) выполняет сложение операндов в двоичной форме в соответствии со следующей таблицей истинности:

1 OR 1 = 1	1 OR 0 = 1
0 OR 1 = 1	0 OR 0 = 0

Результат представляется в десятичной форме счисления.

Упражнение 3. Вычислить результат выполнения выражения $12 \text{ OR } 22$. 12 и 22 занимают в памяти по два байта и в двоичной форме имеют вид 0000000000001100 и 0000000000010110 соответственно. Выполнив сложение согласно таблице истинности, получим двоичное значение суммы 0000000000011110, или 30 в десятичной форме. Следовательно, $12 \text{ OR } 22 = 30$.

Исключающее ИЛИ (XOR) производит сложение операндов в соответствии со следующей таблицей истинности:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Результат преобразуется в десятичную форму счисления.

Упражнение 4. Вычислить результат выполнения выражения $12 \text{ XOR } 22$. 12 и 22 занимают в памяти по два байта и в двоичной форме имеют вид 0000000000001100 и 0000000000010110, соответственно. Выполнив сложение согласно таблице истинности, получим двоичное значение суммы: 0000000000011108, или 26 в десятичной форме. Следовательно, $12 \text{ XOR } 22 = 26$.

Унарная операция сохранения знака (+) оставляет текущий знак числа без изменения. Например:

Выражение	Результат
+(-777)	-777
+(422)	422

Унарная операция изменения знака (-) восстанавливает значение операнда с противоположным знаком. Например:

Выражение	Результат
!-256)	256
-(+39)	-39

Применение операции NOT к данным целочисленных типов вызывает поразрядную инверсию — получение обратного значения соответствующего данному числу двоичного кода. Например:

Выражение	Результат
NOT 0	-1
NOT 7B	-79

Выражения и операции отношения

Выражением отношения называется словосочетание языка, в котором два выражения связаны знаком операции отношения. Выражение отношения оп-

ределает истинность или ложность результата. Операции отношения выполняют сравнение двух операндов и определяют, истинно значение выражения или ложно.

В языке Pascal операции отношения и рассмотренные ниже булевские операции более важны при написании программ, чем в других языках. так как они интенсивно используются для реализации разветвляющихся и циклических алгоритмов. В табл. 3.4 приведены операции отношения, допустимые в версии языка Pascal.

Сравниваемые величины могут принадлежать к любому скалярному или перечисляемому типу данных. Результат всегда имеет булевский тип и принимает одно из двух значений: True (истина) или False (ложь).

Таблица 3.4. Операции отношения

Операция	Название	Выражение	Результат
=	Равно	A=B	True, если A равно B
<>	Не равно	A<>B	True, если A не равно B
>	Больше	A>B	True, если A больше B
<	Меньше	A<B	True, если A меньше B
>=	Больше или равно	A>=B	True, если A больше или равно B
<=	Меньше или равно	A<=B	True, если A меньше или равно B
In	Принадлежность	A in M	True, если A находится в списке M

Логические выражения и операции

Результатом выполнения логического (булевского) выражения является логическое значение True или False. Операндами могут служить только данные булевского типа.

Простейшими видами логических выражений являются следующие:

- логическая константа;
- логическая переменная;
- элемент массива логического типа;
- логическая функция;
- выражение отношения.

Другие логические выражения строятся из вышеперечисленных путем применения логических операций и круглых скобок. Список логических операций приведен в табл. 3.5.

Таблица 3.5. Логические операции

Операция	Действие	Выражение	A	B	Результат
Not	Логическое отрицание	not A	True		False
			False		True

Операция	Действие	Выражение	A	B	Результат
And	Логическое И	A and B	True	True	True
			True	False	False
			False	True	False
			False	False	False
Or	Логическое ИЛИ	A and B	True	True	True
			True	False	True
			False	True	True
			False	False	False
Xor	Исключающее ИЛИ	A xor B	True	True	False
			True	False	True
			False	True	True
			False	False	False

Пример программы, которая сравнивает два вводимых целых числа и печатает значение логического заключения:

```

program Sravnenie;
var
    A,B    integer;
    Test   boolean; {Описание переменной логического типа Test}
begin
    Write('Введите два числа: ');
    Readln(A,B);
    Test := A > B; {Присвоить Test значение результата сравнения}
    Writeln('A больше B - ', Test);
end.
```

Упражнение 5. Введите текст программы, откомпилируйте ее и проверьте результат ее работы для следующих пар значений A и B: 2 и 3, 4 и 2, 5 и 5.

Пример программы, которая вводит два целых числа и выводит на экран результат применения к ним арифметических и логических операций:

```

Program Demo_Operac;
var
    N,M : integer; {Описание двух переменных целого типа integer}
begin
    write('Введите два целых числа');
    Readln(N,M); {Считывание с клавиатуры значений переменных N,M}
    {Выполнение различных логических операций}
    Writeln(N, ' + ', M, ' = ', N + M); {Арифметическое сложение}
    Writeln(N, ' - ', M, ' = ', N - M); {Арифметическое вычитание}
    Writeln(N, ' * ', M, ' = ', N * M); {Арифметическое умножение}
    Writeln(N, ' / ', M, ' = ', N / M); {Вещественное деление}
    Writeln(N, ' div ', M, ' = ', N div M); {Целочисленное деление}
    Writeln(N, ' mod ', M, ' = ', N mod M); {Остаток от деления}
    Writeln('not N = ', not N, ' not M = ', not M); {Отрицание}
    Writeln(N, ' and ', M, ' = ', N and M); {Арифметическое И}
```

```

WriteLn(N, ' or ', M, ' = ', N or M); {Арифметическое ИЛИ}
WriteLn(N, ' xor ', M, ' = ', N xor M); {Исключающее ИЛИ}
WriteLn(N, ' shl ', M, ' = ', N shl M); {Сдвиг влево}
WriteLn(N, ' shr ', M, ' = ', N shr M); {Сдвиг вправо}

```

end.

Упражнение 6. Введите текст программы, запишите ее на диск под именем Demo_Operas, откомпилируйте ее и проверьте результат ее работы для любых пар целых значений N и M.

Операция @. С помощью операции @ можно создать указатель на переменную. В табл. 3.6 показаны операнд и типы результата.

Таблица 3.6. Операция создания указателя

Операция	Действие	Тип операнда	Тип результата
@	Получение указателя	Ссылка на переменную, процедуру или идентификатор функции	Указатель (совместимый с nil)

Операция @ является унарной. В качестве операнда может использоваться ссылка на переменную, процедуру или идентификатор функции. После выполнения операнду возвращается соответствующий указатель, тип которого является таким же, как тип указателя nil, и, следовательно, его можно присвоить любому указателю переменной.

Приоритет операций

Приоритетом называется очередность выполнения операций в выражении. Выполнение каждой операции происходит с учетом ее приоритета. Значения приоритетов указаны в табл. 3.7.

Таблица 3.7. Порядок выполнения операций

Операция	Приоритет	Вид операции
@, not	Первый (высший)	Унарная операция
*, /, div, mod, and	Второй	Операции типа умножения
+, -, or, xor	Третий	Операции типа сложения
=, <>, <, >, <=, >=, in	Четвертый (низший)	Операции отношения

Для определения старшинства операций имеются четыре основных правила.

1. Операнд, находящийся между двумя операциями с различными приоритетами, связывается с операцией, имеющей более высокий приоритет.
2. Операция, находящаяся между двумя операциями с равными приоритетами, связывается с той операцией, которая находится слева.
3. Выражение, заключенное в скобки, перед выполнением вычисляется как отдельный операнд.
4. Операции с равным приоритетом производятся слева направо с возможным регулированием порядка выполнения скобками.

Контрольные вопросы и задания

Вопросы

1. Для чего используется указание типа данных величины?
2. Как описывается тип величины в языке Pascal?
3. Приведите полный перечень типов данных в Turbo Pascal с примерами величин каждого типа.
4. Какие типы данных относят к скалярным типам данных?
5. Охарактеризуйте целочисленные типы данных: какие они могут принимать значения, в каких операциях могут принимать участие, сколько места занимают в памяти.
6. Какие типы отношений определены для данных целого типа? Какие стандартные функции определены для целых чисел?
7. Чем отличаются вещественные числа от целых?
8. Какие функции преобразуют вещественный аргумент в целое число? Чем они отличаются?
9. Охарактеризуйте символьный тип данных.
10. Где применяется булевский тип данных, какие он принимает значения, сколько места требуется для его размещения в памяти?
11. Что такое пользовательские типы данных, чем они отличаются от стандартных типов данных? Приведите примеры данных перечисляемого и интервального типов.
12. Что такое структурированные типы данных?
13. Почему от программиста требуется знание и правильное применение понятия тождественности и совместимости типов величин? Каковы признаки тождественности и условия совместимости типов?
14. Что такое выражение, операция, операнд? Какие операции в языке Pascal вы знаете?
15. Охарактеризуйте каждую арифметическую операцию.
16. Какие операции называются операциями отношения? В чем заключаются особенности результата операций отношения?
17. Охарактеризуйте каждую логическую операцию.
18. Каковы основные правила для определения старшинства операций?

Задания

1. Какие утверждения неправильны:
 - 1) 144 — целое число?
 - 2) 125 — шестнадцатеричное целое число?
 - 3) 124.98 — вещественное число?

- 4) \$1FF — шестнадцатеричное число?
- 5) 'Адрес' — целочисленная константа?
- 6) -12.3 — отрицательное вещественное число?
- 7) 'Сумма' — строковое значение?
2. Какие числа представлены в форме с плавающей точкой:
165, 10.3E+02, 1234.678, 3789, 5.7E0.2, 63.9E-04
3. Какие из следующих утверждений неправильны:
 - 1) для диапазона 1..260 лучше всего подходит тип byte;
 - 2) для диапазона 0..75000 лучше всего подходит тип word;
 - 3) для диапазона 'a'..'z' лучше всего подходит тип char;
 - 4) для вещественных переменных обычно применяется тип real;
 - 5) значение 32000 входит в тип integer.
4. Какой тип подходит для данных диапазона:
6..90?, -40..+45?, +10..+65000?, 100,0..10000,0?
5. Какой идентификатор описывает самый широкий диапазон данных?
6. Какие из следующих соотношений неправильны:
 - 1) $6.22E+02 = 622$;
 - 2) $20E - 03 = 0.02$;
 - 3) $2347.6E-03 = 2.34760$;
 - 4) $0.2E03 = 2000.0$;
 - 5) $1200E+03 = 12000.0$.
7. Какие из следующих утверждений неправильны:
 - 1) перед шестнадцатеричными числами записывается знак #;
 - 2) для описания переменных используется слово var;
 - 3) для описания констант используется слово const;
 - 4) имена переменных не обязательно описывать в разделе var;
 - 5) значение константы можно изменять.
8. Чем отличаются следующие выражения и каков будет результат их выполнения:
 - 1) $10 + 6 * 2 / 2$;
 - 2) $(10 + 6) * 2 / 2$;
 - 3) $(10 + 6 * 2) / 2$;
 - 4) $10 + 6 * (2 / 2)$.
9. Какие приоритеты указаны правильно, а какие нет:
 - 1) приоритет операции * выше, чем + ;
 - 2) not имеет высший приоритет;

- 3) приоритет операции + выше, чем -;
- 4) приоритеты операций * и / одинаковы.
10. Чему равен результат выражения:
 - 1) $61 > 91$?
 - 2) $10 > -7$?
 - 3) $208 > 175$?
11. Какие результаты выполнения выражений неправильны:
 - 1) $(3 > 2) \text{ AND } (5 > 6) = \text{True}$;
 - 2) $(60 > 70) \text{ OR } (100 < 90) = \text{True}$;
 - 3) $('a' < 'b') \text{ XOR } (1 > 0) = \text{True}$;
 - 4) $\text{NOT } (30 > 10) = \text{False}$;
 - 5) $60 \gg 20 = \text{True}$.
12. Какие результаты выполнения выражений неправильны:
 - 1) $24 / 12 = 2$;
 - 2) $11 \text{ div } 5 = 1$;
 - 3) $10 \text{ div } 3 = 3$;
 - 4) $15 + 21 \text{ div } 2 = 25$;
 - 5) $2 \text{ div } 3 = 1$;
 - 6) $6 * 5 = 30$;
 - 7) $11 \bmod 5 = 1$;
 - 8) $14 \bmod (5 + 3) = 2$.
13. Найдите ошибки в программе:


```

program ИМЯ;
  const
    A,B    : integer
    Rezult : byte;
begin
  Write('Введите два числа: ');
  Read(A,B);
  Rezult := A div B;
  Write('Отношение равно ',Rezult)
end
      
```

Исправьте программу и добейтесь компиляции без ошибок.
14. Создайте программу, вычисляющую периметр и площадь треугольника по введенным в диалоге трем сторонам, откомпилируйте и проверьте ее работу.
15. Создайте программу, вычисляющую значение среднего арифметического трех натуральных чисел.

16. Создайте программу, вычисляющую по введенному значению радиуса длину окружности и ее площадь. Для вычисления значения числа $\pi \approx 3,141592...$ в Pascal используется стандартная функция с идентификатором PI.
17. Создайте программу, вычисляющую периметр квадрата по указанному значению его площади.
18. Создайте программу, вычисляющую скорость прямолинейного равномерного движения тела по указанным значениям перемещения и времени, в течение которого это перемещение совершено.
19. Создайте программу подсчета размера оплаты электроэнергии по введенным значениям расхода электроэнергии и тарифа (тариф — стоимость 1 кВт/ч).

Ввод-вывод данных

Общие сведения

Решение самой простой задачи на компьютере не обходится без операций ввода-вывода информации. Ввод данных — это передача информации с внешнего носителя в оперативную память для обработки. Вывод — обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель. Внешним носителем может служить терминал, принтер, дискета, жесткий диск и другие устройства.

В языке Pascal стандартным средством общения человека и компьютера являются стандартные файлы Input и Output, которые по умолчанию являются параметрами программы. Программа получает входные данные из файла Input и помещает результат обработки в файл Output. По умолчанию стандартному файлу Input назначена клавиатура, а файлу Output — экран монитора.

ПРИМЕЧАНИЕ

Как вы узнаете во второй части книги, Delphi обеспечивает возможность программирования ввода-вывода данных с использованием средств GUI.

Процедуры ввода-вывода

Для выполнения операций ввода-вывода служат четыре процедуры: Read, Readln, Write и Writeln. В данном подразделе будет рассмотрено их применение для ввода данных с клавиатуры и вывода на экран и печатающее устройство.

Процедура чтения Read

Процедура чтения Read обеспечивает ввод числовых данных, символов, строк и т. д. для последующей обработки программой. Формат:

Read (X1, X2, ..., Xn);

или

Read (FV, X1, X2, ..., Xn);

где X1, X2, ..., Xn — переменные допустимых типов данных; FV — переменная, связанная с файлом, откуда будет выполняться чтение. В данном подразделе рассматривается в основном первый вариант формата.

Значения X1, X2, ..., Xn вводятся с клавиатуры (в качестве разделителя должен использоваться как минимум один пробел) и отображаются на экране. После ввода данных для одной процедуры Read следует нажать клавишу Enter.

Значения переменных должны вводиться в строгом соответствии с синтаксисом языка Pascal. Если соответствие нарушено (например, X1 имеет тип integer, а при вводе набирается значение типа char), то возникают ошибки ввода-вывода. Сообщение об ошибке имеет вид: I/O error XX, где XX — код ошибки.

Пояснительный текст помогает определить причину программного прерывания.

Пример:

```
var
  I : real;
  J : integer;
  K : char;
begin
  Read (I, J, K);
```

Первый вариант ответа: 212.45 38 'П'

Второй вариант ответа: 'Л' 121.34 23

Первый вариант обеспечивает нормальный ввод данных, так как вводимые значения 212.45 38 и 'П' соответствуют типам переменных I, J, K в процедуре Read. Второй вариант ввода вызовет ошибку с кодом 10, поскольку для переменной I типа real вводится значение типа char.

Если в программе имеется несколько процедур Read, то данные для них вводятся потоком, т. е. после считывания значений переменных для одной процедуры Read данные для следующей процедуры Read вводятся в той же строке, что и для предыдущей, до окончания строки, затем происходит переход на следующую строку.

Пример:

```
var
  A, B, Sum1 : integer;
  C, D, Sum2 : real;
begin
  Read (A, B);
```

```
Sum1 := A + B;
Read (C, D);
Sum2 := C + D;
```

end.

Набираем на клавиатуре: 18758 34 2.62E-02 1.54E+01.

После ввода каждой пары данных нажимаем клавишу Enter, т. е. 18758 34 Enter 2.62E-02 1.54E+01 Enter.

Процедура чтения Readln аналогична процедуре Read, единственное отличие заключается в том, что после считывания последней в списке значения для одной процедуры Readln данные для следующей процедуры Readln будут считываться с начала новой строки. Если в предыдущем примере заменить Read на Readln:

```
...
Readln (A, B);
Sum1 := A + B;
Readln (C, D);
Sum2 := C + D;
```

то после ввода с клавиатуры значений для A и B курсор автоматически перейдет на новую строку, где будут вводиться данные для C и D:

18758 34 Enter
2.62E-02 1.54E+01 Enter

Процедура записи Write

Процедура записи Write производит вывод числовых данных, символов, строк и булевских значений. Формат:

```
Write (Y1, Y2, ..., Yn);
```

или

```
Write (FV, Y1, Y2, ..., Yn);
```

где Y1, Y2, ..., Yn — выражения типа integer, byte, real, char, boolean и т. д.; FV — имя файла, куда производится вывод. Для вывода на принтер FV должно быть равно Lst. Чтобы устройство Lst стало доступным, необходимо подключить модуль Printer с помощью зарезервированного слова uses.

Пример:

```
uses Printer;
var
```

begin

```
Write(234); {Выражение представлено значением}
```

```
Write(A+B - 2); {Выводится результат выражения}
Write(Lst, 'Результат вычислений = ', Result1);
```

end.

В первом варианте формата значения Y1, Y2, ..., Yn выводятся на экран дисплея, во втором — на печатающее устройство.

Форматы вывода

В процедурах вывода Write и Writeln имеется возможность записи выражения, определяющего ширину поля вывода. В приведенных ниже форматах используются следующие обозначения:

I, p, q — целочисленное выражение;

R — выражение вещественного типа;

B — выражение булевского типа;

Ch — выражение символьного типа;

S — выражение строкового типа;

— цифра;

* — знак «+» или «-»;

_ — пробел.

I — выводится десятичное представление величины I, начиная с позиции расположения курсора.

Значение I	Выражение	Результат
134	Write (I);	134
5671	Write (I);	5671
287	Write (I.I.I);	287287287

I:p — выводится десятичное представление величины I в крайние правые позиции поля шириной p.

Значение I	Выражение	Результат
134	Write (I:6);	___134
1	Write (I:10);	_____1
312	Write (I+I:7);	_____624

R — в поле шириной 18 символов выводится десятичное представление величины R в формате с плавающей точкой. Если $R \geq 0.0$, используется формат #.#####E*##.

Если $R < 0.0$, формат имеет вид — #.#####E*##.

Значение R	Выражение	Результат
715.432	Write (R);	__7.1543200000E+02
-1.919E+01	Write (R);	__-1.919000000E+01
567.986	Write (R/2);	__2.839930000E+02

R:p — в крайние правые позиции поля шириной p символов выводится десятичное представление значения R в формате с плавающей точкой.

Если $R \geq 0.0$, используется формат `_. . . _##. . #E*##`, причем минимальная ширина поля вывода составляет 7 символов.

Если $R < 0.0$, формат имеет вид `_. . . _ — #. ##. . #E*##`. Минимальная ширина поля вывода 8 символов. После десятичной точки выводится по крайней мере одна цифра.

Значение R	Выражение	Результат
511.04	Write (R:15):	5.110400000E+02
-511.04	Write (R:15):	-5.110400000E+02
46.7B	Write (-R:12):	-4.67B00E+01

R:p;q — в крайние правые позиции поля шириной *p* символов выводится десятичное представление значения *R* в формате с фиксированной точкой, причем после десятичной точки выводится *q* цифр ($0 \leq q \leq 24$), представляющих дробную часть числа. Если $q = 0$, то ни дробная часть, ни десятичная точка не выводятся. Если $q > 24$, то при выводе используется формат с плавающей точкой.

Значение R	Выражение	Результат
511.04	Write (R:B:4):	511.0400
-46.7B	Write (R:7:2):	—46.7B
-46.7B	Write (R:9:4):	—46.7B00

Ch — начиная с позиции курсора выводится значение *Ch*.

Значение Ch	Выражение	Результат
'X'	Write (Ch):	X
'Y'	Write (Ch):	Y
'!'	Write (Ch, Ch, Ch):	!!!

Ch:p — в крайнюю правую позицию поля шириной *p* выводится значение *Ch*.

Значение Ch	Выражение	Результат
'X'	Write (Ch:3):	—X
'Y'	Write (Ch:5):	—Y
'!'	Write (Ch:2,Ch:4):	—!—!

S — начиная с позиции курсора, выводится значение *S* (строка или массив символов, если его длина соответствует длине строки).

Значение S	Выражение	Результат
'Day N'	Write (S):	Day N
'Ведомость 11'	Write (S):	Ведомость 11
'RRRDDD'	Write (S,S):	RRRDDDRRDD

S:p — значение *S* выводится в крайние правые позиции поля шириной *p* символов.

Значение S	Выражение	Результат
'Day N'	Write (S:10):	—Day N
'Ведомость 11'	Write (S:13):	—Ведомость 11
'RRRDDD'	Write (S:7,S:7):	—RRRDDD_RRRDDD

B — выводится результат выражения *B* True или False, начиная с текущей позиции курсора.

Значение B	Выражение	Результат
True	Write (B):	True
False	Write (B, not B):	False True

B:p — в крайние правые позиции поля шириной *p* символов выводится результат булевого выражения *B* True или False.

Значение B	Выражение	Результат
True	Write (B:6):	—True
False	Write (B:10):	—False
True	Write (B:5,not B:7):	—True—False

Оператор записи **Writeln** аналогичен процедуре **Write**, но после вывода последнего в списке значения для текущей процедуры **Writeln** происходит перевод курсора к началу следующей строки.

Процедура **Writeln**, записанная без параметров, вызывает перевод строки.

Для пояснения работы процедуры **Writeln** приведем фрагмент программы:

```
A := 4;
B := 6;
C := 55;
Write(A:3); Write(B:3); Write(C:3);
Sum:= A + B + C;
Writeln('A=', A);
Writeln('B=', B);
Writeln('C=', C);
Writeln('Сумма A+B+C равна ', Sum);
```

Результат выполнения:

```
4 6 55
A=4
B=6
C=55
Сумма A+B+C равна 65
```

Примером использования формата в процедуре **Writeln** может служить следующая программа.

```
Program DemoWriteln; {Программа вычисляет площадь прямоугольника и выводит
на печать результат}
uses Printer; {Подключение модуля с библиотекой процедур и функций,
обслуживающих принтер — устройство Lst}
var
  A, B, S: integer; {A,B — длина сторон, S — площадь}
begin
  A:= 8; B:= 4;
  S:= A * B;
```

```

Writeln(Lst, '-----');
Writeln(Lst, '   Сторона А   Сторона В   Площадь   ');
Writeln(Lst, '-----');
Writeln(Lst, ' ', A:7, B:11, S:11, ' ':5);
Writeln(Lst, '-----');
end.

```

В результате работы программы получим таблицу, напечатанную на бумаге:

Сторона А	Сторона В	Площадь
6	4	32

каждая строка которой будет печататься с первой позиции новой строки печатающего устройства.

Упражнение. Загрузите интегрированную среду программирования, введите текст данной программы, откомпилируйте ее и проверьте результат выполнения. Если к компьютеру не подключен принтер, то проверьте вывод на экран, для чего в записи списка вывода процедуры Writeln опустите слово Lst.

Контрольные вопросы и задания

Вопросы

1. Какие процедуры служат в Pascal для выполнения операций ввода-вывода?
2. В чем заключается отличие процедуры чтения Readln от процедуры Read?
3. Как задать вывод информации на принтер?
4. Какой стандартный модуль обеспечивает использование принтера в вашей программе? Каким образом описывается его подключение?
5. Для чего в процедурах вывода Write и Writeln определяется ширина поля вывода? Какие обозначения используются в форматах вывода?

Задания

1. Создайте программу, которая, используя процедуру Writeln, изображает на экране домик:

```

      *
    * *
  *   *
*****
  *   *
  *   *
*****

```

2. Составьте программу, которая выводит на экран компьютера заставку, идентичную следующей:

```

+-----+ +*****+
          ЭММ          *
          Суммы чисел   *
          Грив В.И.     *
+-----+ +*****+

```

3. Напишите программу, которая вводит значения трех переменных: А, В, С и выводит их сумму. Ввод каждого значения следует производить с новой строки. Результат также помещается на отдельную строку. При составлении программы обеспечьте приглашение к вводу данных.
4. Напишите программу, которая вводит значения четырех переменных А, В, С, D типа integer и выводит их сумму. Ввод пары значений А и В следует произвести на одной строке, С и D — на другой. Результат следует вывести на отдельную строку, и курсор оставить на той же строке.
5. Напишите программу, которая вводит значения двух переменных: А, В типа integer с приглашениями к вводу каждой переменной и выводит их сумму. Результат ввода и результат расчета требуется вывести на экран параллельно на печать. Приглашение и ввод каждого значения следует произвести в отдельных строках. Вывод сопроводите пояснением.
6. Напишите программу ввода значений А, В, С в одной строке и выведите результат вычисления выражения $A * (B / 3,14) + (C * 3)$ в отдельной строке.
7. Напишите программу ввода значений R, Y в одной строке и выведите результат вычисления выражения $R * Y^2 + (Y / 5)$ в той же строке.
8. Напишите программу вычисления площади прямоугольного треугольника. Значения катетов которого А и В вводятся с клавиатуры. Результат требуется вывести в следующем виде: «Для значений катетов 4 и 6 площадь прямоугольного треугольника равна 12».
9. Напишите программу вычисления идеального веса человека по формуле: $W = (R - 100) * 70$. Рост в см — 100. Значение роста вводится с клавиатуры. Результат требуется вывести в следующем виде: «Для человека ростом 165 см идеальный вес равен 65 кг».
10. Напишите программу получения следующей формы:

```

+*****+
          Е * А/В *
+-----+ +*****+
          4
          * * * * *

```

11. Вы положили деньги в банк на срочный депозит на три месяца из расчета 60% годовых. Напишите программу, которая вычислит причитающуюся вам через три месяца сумму.

12. Розничная цена мужского костюма составляет P руб. Торговая скидка в пользу магазина составляет $T\%$ от розничной цены. Создайте программу определения оптовой цены костюма.
13. Создайте программу расчета масс соли и воды, требующихся для приготовления раствора массой m г с массовой долей $w\%$.
14. Создайте программу исследования положительного вещественного числа A , в которой определялись бы значения следующих величин: целая часть, дробная часть, значение арифметического квадратного корня, остаток от деления на 5.
15. Создайте программу, определяющую, сколько времени в минутах затратит школьник на дорогу от школы до стадиона, если известна длина этого расстояния S и средняя скорость движения школьника V км/ч? Значения S и V вводятся с клавиатуры.
16. Создайте программу, вычисляющую, сколько процентов от $A + B - C$ приходится:
 - 1) на A ;
 - 2) на B ;
 - 3) на C .

Глава 4

Операторы

Общие сведения

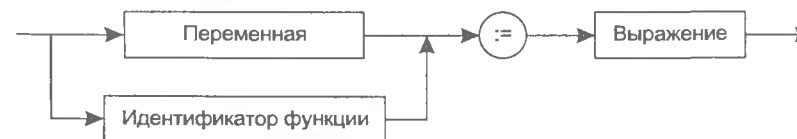
Оператором называется предложение языка программирования, задающее полное описание некоторого действия, которое необходимо выполнить. Основная часть программы на языке Turbo Pascal представляет собой последовательность операторов. Разделителем операторов служит точка с запятой. Все операторы языка Turbo Pascal можно разделить на две группы: простые и структурные.

Простые операторы

Операторы, не содержащие других операторов, называются *простыми*. К ним относятся операторы присваивания, безусловного перехода, вызова процедуры и пустой оператор.

Оператор присваивания

Оператор присваивания ($:=$) предписывает выполнить выражение, заданное в его правой части, и присвоить результат переменной, идентификатор которой расположен в левой части. Переменная и выражение должны быть совместимы по типу. Ниже представлен общий вид оператора присваивания.



Оператор присваивания выполняется следующим образом: сначала вычисляется выражение в правой части присваивания, а затем его значение присваивается переменной, указанной в левой части оператора. Например, для

оператора `Rezult := A div B`; сначала выполняется целочисленное деление значения переменной `A` на значение переменной `B`, а затем результат присваивается переменной `Rezult`.

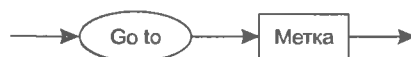
В предыдущих примерах мы использовали этот оператор, например:

```
A := B;
S := A * B;
Ostatok := A mod B;
Ratio := A / B;
```

Оператор безусловного перехода (go to)

Оператор безусловного перехода (`go to`) означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой оператор, отмеченный меткой.

Синтаксическая диаграмма для оператора перехода имеет следующий вид:



Напомним, что метка объявляется в разделе описания меток и может содержать как цифровые, так и буквенные символы.

Пример:

```
go to 999;
go to EndBlock;
```

ВНИМАНИЕ

При использовании оператора `go to` необходимо помнить, что областью действия метки является только тот блок, в котором она описана. Передача управления в другой блок запрещена.

Использование безусловной передачи управления в программе считается теоретически избыточным и подвергается серьезной критике, так как способствует созданию малопонятных и запутанных программ, которые вызывают большие сложности при отладке и сопровождении. Поэтому рекомендуется минимальное использование оператора `go to` с соблюдением следующих правил:

- следует стремиться применять операторы перехода (если кажется невозможным обойтись без них) для передачи управления только вниз (вперед) по тексту программы; при необходимости передачи управления назад следует использовать операторы цикла;
- расстояние между меткой и оператором перехода на нее не должно превышать одной страницы текста (или высоты экрана дисплея).

Оператор вызова процедуры

Оператор вызова процедуры служит для активизации предварительно определенной пользователем, или стандартной, процедуры, например:

```
{Вызов стандартной процедуры очистки экрана}
{Вызов пользовательской процедуры}
```

Пустой оператор

Пустой оператор не содержит каких-либо символов и не выполняет каких-либо действий. Обычно пустой оператор используется для организации перехода к концу локального или глобального блока в случаях, если необходимо пропустить несколько операторов, но не выходить из блока. Для этого перед зарезервированным словом `end` ставятся метка и двосточие, например:

```
{Переход в конец блока}
```

```
{Оператор помечен меткой}
```

Структурные операторы

Структурные операторы представляют собой конструкции, построенные из других операторов по строго определенным правилам. Все структурные операторы можно разделить на три группы: составные, условные, повтора.

Составной оператор

Составной оператор представляет собой группу из произвольного числа операторов, разделенных друг от друга точкой с запятой, и ограниченную операторами `begin` и `end`.

Синтаксическая диаграмма составного оператора записывается так:



Составной оператор воспринимается как единое целое и может находиться в любом месте программы, где синтаксис языка допускает наличие оператора.

Условные операторы

Условные операторы предназначены для выбора на исполнение одного из возможных действий (операторов) в зависимости от некоторого условия (при этом одно из действий может быть пустым, т. е. отсутствовать). В качестве условий выбора используется значение логического выражения. В Turbo Pascal имеются два условных оператора: if и case.

Оператор условия if

Оператор условия if является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Синтаксическая диаграмма оператора условия if выглядит следующим образом:



Как видно из диаграммы, он может принимать одну из следующих форм:

if <условие> then <оператор1>	ЕСЛИ <условие> ТО <оператор1>
else <оператор2>;	ИНАЧЕ <оператор2>
if <условие> then <оператор>;	ЕСЛИ <условие> ТО <оператор>

Оператор условия if выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисления получается значение булевского типа. В первом случае, если значение выражения равно True (истина), выполняется <оператор1>, указанный после слова then (ТО). Если результат вычисления выражения в условии равен False (ложь), то выполняется <оператор2>. Во втором случае, если результат выражения равен True, то выполняется <оператор>, если False — оператор, следующий сразу за оператором if. Операторы if могут быть вложенными.

Пример фрагмента программы с оператором условия if:

```
Read(Ch);
if Ch='N' then Parol:= True
    else Parol:= False;

Read(X);
if Parol = True then
    if X = 100 then Write('Пароль и код правильные')
        else begin
            Writeln('Ошибка в коде');
            Halt(1)
        end;
end;
```

В данном примере с клавиатуры считывается значение переменной символьного типа Ch. Затем проверяется условие Ch = 'N'. Если оно выполняется, то пере-

менной Parol булевского типа присваивается значение True, если условие не выполняется — False. Затем с клавиатуры считывается значение кода X. Затем оператор if проверяет условие Parol = True. Если оно имеет значение True, то выполняется проверка введенного пароля оператором if X=100. Если условие X=100 имеет значение True, то выводится сообщение «Пароль и код правильные», и управление в программе передается на оператор, следующий за словом end, если оно имеет значение False, то выполняется составной оператор, стоящий после слова else, который выводит на экран сообщение «Ошибка в коде» и вызывает стандартную процедуру Halt(1) для остановки программы.

ПРИМЕЧАНИЕ

При использовании вложенных условных операторов может возникнуть синтаксическая неоднозначность, иллюстрируемая следующей схемой:
if условие1 then if условие2 then <оператор1> else <оператор2>
Возникающая двусмысленность, к какому оператору if принадлежит часть else <оператор2>, разрешается тем, что служебное слово else всегда ассоциируется (связывается) с ближайшим по тексту служебным словом if, которое еще не связано со служебным словом else.
В связи с этим следует проявлять аккуратность при записи вложенных операторов условия.

Упражнение 1. Составим программу, которая вычисляет частное двух целых чисел. В связи с тем что делить на ноль нельзя, организуем контроль ввода данных. Для контроля вводимых значений делителя используем оператор условного перехода if...then...else. Текст программы будет выглядеть следующим образом:

```
program Tutor_If_Then_Else;
var
  A,B : integer;
  Result: real;
begin
  Write('Введите значение делимого A >');
  Read(A);
  Write('Введите значение делителя B >');
  Read(B);
  if B=0
    {Контроль ввода}
  then Writeln('На ноль делить нельзя') {Условие выполнено}
  else {Условие не выполнено}
    begin {Начало составного оператора}
      Result := A / B;
      Writeln('Частное чисел '.A.' и '.B.' = '.Result);
    end; {Конец составного оператора}
end;
```

Введите текст программы, откомпилируйте ее и исполните для разных целых значений переменных A и B. Попробуйте задать значение B = 0 и убедитесь, что контроль ввода работает. В будущих ваших программах выполняйте контроль ввода данных.

Оператор выбора case

Если один оператор if может обеспечить выбор из двух альтернатив, то оператор выбора case позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором* (selection — выбор альтернативы), и списка параметров, каждому из которых предшествует список констант выбора (список может состоять и из одной константы). Синтаксическая диаграмма оператора case выглядит следующим образом:



Следуя данной диаграмме, получим следующий формат записи оператора case:

```

case <выражение-селектор> of
  <список1>: <оператор1>;
  <список2>: <оператор2>;

  <списокN>: <операторN>;
else <оператор>;
end;
```

Оператор case работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом else. Если слово else отсутствует, то активизируется оператор, находящийся за словом end, т. е. первый оператор за границей case.

Селектор должен относиться к одному из целочисленных типов (со значениями в диапазоне -32768..32767): булевскому, литерному или пользовательскому. Список констант выбора состоит из произвольного количества значений, или диапазонов, отделенных друг от друга запятыми. Границы диапазона записываются двумя константами через разграничитель «..». Тип констант в любом случае должен совпадать с типом селектора. В приведенном выше синтаксическом описании предполагается использование одного оператора для каждой альтернативы, но при необходимости можно задать несколько операторов, сгруппировав их в составной оператор. В то же время ветвь else допускает использование последовательности операторов, разделенных символом «;».

При использовании оператора выбора case должны выполняться следующие правила.

1. Значения выражения «переключателя», записанного после служебного слова case, должны принадлежать дискретному типу (лат. discretus — прерывистый, дробный, состоящий из отдельных частей); для целого типа они должны лежать в диапазоне integer.

2. Все константы, предшествующие операторам альтернатив, должны иметь тип, совместимый с типом выражения.
3. Все константы в альтернативах должны быть уникальны в пределах оператора варианта (т. е. повторение констант в альтернативах не допускается); диапазоны не должны пересекаться и не должны содержать констант, указанных в данной или других альтернативах.

Ниже приведены типичные формы записи оператора case.

Селектор интервального типа:

```

case I of
  1..10 : Writeln ('число ', I:4, ' в диапазоне 1 - 10');
  11..20 : Writeln ('число ', I:4, ' в диапазоне 11 - 20');
  21..30 : Writeln ('число ', I:4, ' в диапазоне 21 - 30')
        else Writeln ('число ', I:4, ' вне пределов контроля')
end;
```

Селектор целочисленного типа:

```

case I of
  1 : Z := I + 10;
  2 : Z := I + 100;
  3 : Z := I + 1000
end;
```

Селектор перечисляемого пользовательского типа:

```

var
  Season: (Winter, Spring, Summer, Autumn);
begin
  :
  case Season of
    Winter : Writeln('Winter');
    Spring : Writeln('Spring');
    Summer : Writeln('Summer');
    Autumn : Writeln('Autumn')
  end;
end;
```

Пример программы с использованием оператора case, которая по введенному номеру дня недели выводит на экран его название на русском языке.

```

Program Day_Week;
var Day : byte;
begin
  Write ('Введите номер дня недели :');
  Readln(Day);
  case Day of {Вычисление значения селектора и выбор}
    1 : Writeln('Понедельник');
    2 : Writeln('Вторник');
    3 : Writeln('Среда');
```

```

4 . Writeln('Четверг');
5 . Writeln('Пятница');
6 . Writeln('Суббота');
else
    Writeln('Воскресенье');
end;
end.

```

В данном примере на экран выводится приглашение «Введите номер дня недели :», с клавиатуры считывается целочисленное значение дня недели и присваивается переменной Day. Затем, в зависимости от значения селектора Day, обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Например, если значение Day равно 3, то реализуется оператор Writeln('Среда'). Если значение Day равно 7, а ни одна из констант не равна этому значению селектора, то выполняется оператор, стоящий за словом else (на экран выводится текст «Воскресенье»). Если слово else отсутствует, то активизируется оператор, находящийся за словом end;, т. е. первый оператор за границей case. Если значение Day не равно значению ни одной из констант выбора (например, Day = 8 или Day = 0), то активизируется оператор, находящийся за словом end;, т. е. первый оператор за границей case — оператор end.

Упражнение 2. Введите текст программы, откомпилируйте ее и исполните для разных целых значений переменной Day. Проанализируйте действие оператора выбора case при разных значениях номера дня (Day). Обратите внимание, что Day и константы выбора 1–6 имеют одинаковый тип — byte.

Упражнение 3. Модифицируйте программу, дополнив ее контролем правильности ввода (если пользователь введет значение дня недели <1 или >7, то необходимо вывести на экран сообщение об ошибке ввода). Для этого используйте оператор условия if...then...else.

Операторы повтора

Если в программе возникает необходимость неоднократно выполнить некоторые операторы, то используются операторы повтора (цикла). В языке Pascal различают три вида операторов цикла: while, repeat, for. Они используются для организации циклов различных типов. Выражение, управляющее повторениями, должно иметь булевский тип.

Если число повторений оператора (составного оператора) заранее неизвестно, а задано лишь условие его повторения (или окончания), используются операторы while, repeat. Оператор for используется, если число повторений известно заранее.

Оператор while

Оператор while (пока) часто называют *оператором цикла с предусловием* за то, что проверка условия выполнения тела цикла производится в самом на-

чале оператора. Синтаксическая диаграмма для данного оператора выглядит следующим образом:



Формат записи:

```

while <условие продолжения повторений> do
    <тело цикла>;

```

Условие — булевское выражение, тело цикла — простой или составной оператор. Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен True, то тело цикла выполняется, и снова вычисляется выражение условия. Если результат равен False, происходят выход из цикла и переход к первому после while оператору.

Примером работы while может служить программа DemoWhile, которая производит суммирование десяти произвольно введенных целых чисел.

```

program DemoWhile;
const
    Limit = 10; {Ограничение на количество вводимых чисел}
var
    Count, Item, Sum: integer;
begin
    Count := 0;      {Счетчик чисел}
    Sum := 0;        {Сумма чисел}
    while (Count < Limit) do {Условие выполнения цикла}
    begin
        Count := Count + 1;
        Write('Введите ', Count, '-е целое число: ');
        Readln(Item); {Ввод очередного числа с клавиатуры}
        Sum := Sum + Item;
    end;
    Writeln('Сумма введенных чисел равна ', Sum);
end.

```

В данном примере в разделе описания констант описана константа Limit = 10, задающая ограничение на количество вводимых чисел. В разделе описания переменных описаны переменные Count, Item, Sum целочисленного типа.

В начале выполнения программы обнуляются значения счетчика введенных чисел Count и их суммы.

Затем выполняются цикл ввода 10 чисел и их суммирование. Вначале оператор условия while проверяет условие Count < Limit. Если выражение истинно, то выполняется составной оператор в теле цикла:

```

begin
    Count := Count + 1;
    Write('Введите ', Count, '-е целое число: ');

```

```

Readln(Item);
m:= Sum+Item;
d

```

в котором вводится значение очередного числа, и на это значение увеличивается значение суммы. После этого управление в программе вновь передается оператору цикла `while`, опять проверяется условие `Count < Limit`. Если выражение истинно, то выполняется составной оператор и т. д., пока значение переменной `Count` меньше 10.

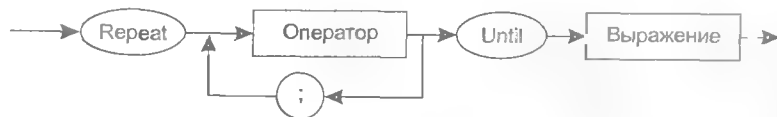
Как только значение `Count` станет равно 10 и условие `Count < Limit` не будет соблюдено, выполнение цикла завершится, а управление будет передано на оператор, находящийся за словом `end`, т. е. первый оператор за границей `while`. Это вызов процедуры `Writeln`, которая выведет сообщение «Сумма введенных чисел равна» и напечатает значение переменной `Sum`.

Упражнение 4. Введите текст программы, откомпилируйте ее и исполните для разных значений чисел. Проверьте выполнение программы, отслеживая значения переменных `Sum`, `Item` и условия `Count < Limit` в режиме пошагового выполнения.

Оператор повтора repeat

Оператор повтора `repeat` аналогичен оператору `while`, но отличается от него, во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла (очередной итерации) и таким образом гарантируется хотя бы однократное выполнение цикла, а во-вторых, тем, что критерием прекращения цикла является равенство выражения константе `True`. За это цикл `repeat` часто называют *циклом с постусловием*, или циклом «ДО», так как он прекращает выполняться, как только значение выражения условия, записанного после слова `until`, становится равным `True` (истина).

Оператор повтора `repeat` состоит из заголовка `repeat`, тела и условия окончания `until`. Синтаксическая диаграмма для данного оператора выглядит следующим образом:



Формат записи:

```

repeat
  <оператор>
.
  <оператор>
until <условие окончания цикла>

```

Операторы, заключенные между словами `repeat` и `until`, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода

из цикла. Именно поэтому цикл, организованный с помощью оператора `repeat`, в любом случае выполнится хотя бы один раз. Если результат булевского выражения равен `False`, то тело цикла активизируется еще раз; если результат равен `True`, то происходит выход из цикла.

ВНИМАНИЕ

При программировании операторов тела цикла следует обеспечить влияние по крайней мере одного из операторов тела цикла на значение условия, иначе цикл будет выполняться бесконечно.

В следующем фрагменте показано, как оператор `repeat` используется для ожидания нажатия клавиш `Y` и `N`. Нажатие других клавиш будет игнорироваться:

```

uses Crt;
var
  YN: char;
begin
  ...
  repeat
    YN:= ReadKey
  until Ucase(YN) in ['Y','N'];

end.

```

Примером действия оператора `repeat` может служить программа `DemoRepeat`, которая вводит и суммирует любое количество целочисленных значений. Если введено значение 999, то на экран выводится результат суммирования.

```

program DemoRepeat;
var
  X: integer;
  Sum: real;
begin
  Sum:=0;
  repeat
    Write('Значение X= '); {Повторять}
    Readln(X); {Начало тела цикла}
    {Считать очередное значение X с клавиатуры}
    if X <> 999 then
      Sum:= Sum+X;
  until X = 999; {Условие окончания цикла}
  Writeln('Сумма введенных чисел= ',Sum);
end.

```

В данном примере в разделе описания переменных описана переменная `X` целочисленного типа `integer` и `Sum` вещественного типа `real`.

В начале выполнения программы обнуляется значение суммы чисел. Затем при помощи зарезервированного слова `repeat` объявляется цикл, после чего следуют операторы тела цикла, которые выводят на экран запрос «Значение `X=`» и считывают введенное с клавиатуры значение `X`. Оператор `if` проверяет

его на неравенство числу 999 и, если оно не равно 999, увеличивает значение суммы Sum на значение числа X. В конце цикла оператор `until X = 999` проверяет условие окончания цикла. Если значение выражения $X = 999$ истинно, то цикл завершится, а управление в программе будет передано на оператор, находящийся за словом `until`, т. е. первый оператор за границей цикла `repeat`. Это вызов процедуры `Writeln`, которая выведет сообщение «Сумма введенных чисел равна» и напечатает значение переменной Sum.

Упражнение 5. Введите текст программы, откомпилируйте ее и выполните для разных значений чисел. Проверьте выполнение программы, отслеживая значения переменных X, Sum и выполнение условия $X = 999$ в режиме пошагового выполнения.

Оператор повтора for

В случаях когда число повторений заранее известно, для организации циклической обработки информации применяется оператор повтора `for`. Часто этот оператор повтора называют *оператором цикла с параметром*, так как число повторений задается переменной, называемой *параметром цикла*, или *управляющей переменной*.

Оператор повтора `for` состоит из заголовка и тела цикла. Синтаксическая диаграмма для данного оператора выглядит следующим образом:



Как видно из диаграммы, этот оператор может быть представлен в двух форматах:

```
for <параметр цикла> := <S1> to <S2> do <оператор>;
for <параметр цикла> := <S1> downto <S2> do <оператор>;
```

где S1 и S2 — выражения, определяющие соответственно начальное и конечное значения параметра цикла;

for ... do — заголовок цикла;
<оператор> — тело цикла.

Тело цикла может являться простым или составным оператором. Оператор `for` обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметра цикла от начального до конечного.

Заголовок оператора повтора `for` определяет:

- диапазон изменения значений управляющей переменной (параметра цикла) и одновременно число повторений оператора, содержащегося в теле цикла;

- направление изменения значения параметра цикла (возрастание — `to` или убывание — `downto`).

Пример:

```
for I:= 1 to 100 do Read(M[I]);    {Чтение элементов массива}
for I:= 100 downto 1 do Write(M[I]); {Вывод элементов массива}
```

При первом обращении к оператору `for` вначале вычисляются выражения S1, S2 и осуществляется присваивание <параметр цикла>:=S1.

После этого циклически повторяются следующие действия.

1. Проверяется условие <параметр цикла>:=S2.
2. Если условие выполнено, то оператор `for` продолжает работу (выполняется оператор в теле цикла), если условие <параметр цикла>:=S2 не выполнено, то оператор `for` завершает работу, и управление в программе передается на оператор, следующий за циклом.
3. Значение управляющей переменной изменяется на +1 (`to`) или -1 (`downto`) и далее с п. 1. Обратите внимание, что шаг изменения управляющей переменной — единица.

На использование управляющей переменной (параметра цикла) в цикле `for` накладываются следующие ограничения.

1. В качестве параметра должна использоваться простая переменная, описанная в текущем блоке.
2. Управляющая переменная должна иметь дискретный тип.
3. Начальные и конечные значения диапазона должны иметь тип, совместимый с типом управляющей переменной. При этом допустим любой скалярный тип, кроме вещественного.
4. В теле цикла запрещается явное изменение значения управляющей переменной (например, при помощи оператора присваивания).
5. После завершения оператора значение управляющей переменной становится неопределенным, если только выполнение оператора не было прервано оператором перехода.

Примером действия оператора `for` может служить программа `DemoFor`, которая выводит на экран таблицу перевода из градусов по шкале Цельсия (C) в градусы по Фаренгейту (F) для значений от 15 до 30 °C с шагом 1 градус. Перевод осуществляется по формуле: $F = C * 1.8 + 32$.

program DemoFor:

var

I: integer;

F: real;

begin

Writeln('Температура');

```
for I:= 15 to 30 do {Заголовок цикла с параметром}
begin              {Начало тела цикла}
```

```

F:= I*1.8+32;
writeln('По Цельсию= ',I,' по Фаренгейту= ', F:5:2);
end;

```

end

В блоке описания переменных описаны параметр цикла `I` типа `integer` и переменная `F` — температура по Фаренгейту типа `real`. Переменная `I`, помимо функций управляющей переменной, является переменной, хранящей целочисленные значения температуры по шкале Цельсия. В начале выполнения программы на экран выводится надпись «Температура», а затем при помощи оператора повтора выводится таблица соотношения температуры в шкалах Цельсия и Фаренгейта. Печать таблицы выполняется оператором

```

writeln('По Цельсию= ',I,' по Фаренгейту= ', F:5:2);

```

Цикл выполняется следующим образом.

При первом обращении к оператору `for` вычисляются значения начального (15) и конечного (30) параметров цикла, и управляющей переменной `I` присваивается начальное значение 15.

Затем циклически выполняются следующие действия:

1. Проверяется условие `I<=30`.
2. Если оно соблюдается, то выполняется составной оператор в теле цикла, т. е. рассчитывается значение выражения `I*1.8+32`, затем оно присваивается переменной `F`, и на экран выводится сообщение:

```

'По Цельсию= ',I,' по Фаренгейту= ', F:5:2

```
3. Если условие `I<=30` не соблюдается, т. е. как только `I` станет больше 30, оператор тела цикла не выполняется, а управление в программе передается за пределы оператора `for`, в нашем примере на оператор `end`; Программа завершает работу.
4. Значение параметра цикла `I` увеличивается на единицу, и управление передается в заголовок цикла `for` для проверки условия.

Далее цикл повторяется с п. 1.

Вложенные операторы цикла

Если телом цикла является циклическая структура, то такие циклы называют *вложенными*. Цикл, содержащий другой цикл, называют *внешним*, а цикл, содержащийся в теле другого цикла, называют *внутренним*. Внешний и внутренние циклы могут быть трех видов: циклами с предусловием `while`, циклами с постусловием `repeat` или циклами с параметром `for`.

ВНИМАНИЕ

Правила организации внешнего и внутреннего циклов такие же, как и для простого цикла каждого из видов. Но при программировании вложенных циклов необходимо соблюдать следующее дополнительное условие: все операторы внутреннего цикла должны полностью располагаться в теле внешнего цикла.

Рассмотрим пример простой задачи, решение которой предполагает использование вложенных циклов, — задачи вывода на экран таблицы умножения. С использованием цикла `for` вариант решения данной задачи может быть следующим:

```

for I:=1 to 10 do
  for J:=1 to 10 do
    writeln(I,' * ',J,' = ',I*J);
  end;
end;

```

end

Проанализируем работу данной программы. В разделе описания переменных описываются переменные `I`, `J` целого типа `byte`, выполняющие функции управляющих переменных циклов `for`.

Выполнение программы начинается с внешнего цикла. При первом обращении к оператору внешнего цикла `for` вычисляются значения начального (1) и конечного (10) параметров цикла и управляющей переменной `I` присваивается начальное значение 1.

Затем циклически выполняются следующие действия.

1. Проверяется условие `I<=10`.
2. Если оно соблюдается, то выполняется оператор в теле цикла, т. е. выполняется внутренний цикл.
3. При первом обращении к оператору внутреннего цикла `for` вычисляются значения начального (1) и конечного (10) параметров цикла и управляющей переменной `J` присваивается начальное значение 1.

Затем циклически выполняются следующие действия.

1. Проверяется условие `J<=10`.
2. Если оно удовлетворяется, то выполняется оператор в теле цикла, т. е. оператор `writeln(I,' * ',J,' = ',I*J)`, выводящий на экран строку таблицы умножения в соответствии с текущими значениями переменных `I` и `J`.
3. Затем значение управляющей внутреннего цикла `J` увеличивается на единицу, и оператор внутреннего цикла `for` проверяет условие `J<=10`. Если условие соблюдается, то выполняется тело внутреннего цикла при неизменном значении управляющей переменной внешнего цикла до тех пор, пока выполняется условие `J<=10`.

Если условие `J<=10` не удовлетворяется, т. е. как только `J` станет больше 10, оператор тела цикла не выполняется, внутренний цикл завершается и управление в программе передается за пределы оператора `for` внутреннего цикла, т. е. на оператор `for` внешнего цикла.

4 Значение параметра цикла I увеличивается на единицу, и проверяется условие $I \leq 10$. Если условие $I \leq 10$ не соблюдается, т. е. как только I станет больше 10, оператор тела цикла не выполняется, внешний цикл завершается и управление в программе передается за пределы оператора `for` внешнего цикла, т. е. на оператор `end`, и программа завершает работу.

Таким образом, на примере печати таблицы умножения наглядно видно, что при вложении циклов изменение управляющей переменной вложенного цикла проходит полный цикл от начального до конечного значения при неизменном значении управляющей переменной внешнего цикла, затем значение управляющей переменной внешнего цикла изменяется на единицу и опять изменение параметра внутреннего цикла претерпевает изменения от начального до конечного значения с шагом, равным единице. И так до тех пор, пока значение параметра внешнего цикла не станет больше конечного значения, определенного в операторе `for` внешнего цикла.

Упражнение 6. Введите текст программы `Tab_Umn1`, откомпилируйте и исполните ее в режиме пошагового выполнения. Наблюдая за значениями параметров циклов I и J и выражений $I \leq 10$ и $J \leq 10$, проанализируйте действие операторов внешнего и внутреннего циклов.

Упражнение 7. Изучите текст программы `Tab_Umn2`, объясните, чем он отличается от предыдущей программы. Объясните формат вывода значения произведения $I * J$ на экран оператором `Write(I*J:4)`. Каков будет результат выполнения программы?

```
program Tab_Umn2;
var
  I, J : byte;
begin
  for I:=1 to 10 do           {Внешний цикл}
  begin
    for J:=1 to 10 do         {Внутренний цикл}
    Write(I*J:4);
    WriteLn;
  end;
end.
```

Введите текст программы `Tab_Umn2`, откомпилируйте и исполните ее в режиме пошагового выполнения. Наблюдая за значениями параметров циклов I и J и выражений $I \leq 10$ и $J \leq 10$, проанализируйте действие операторов внешнего и внутреннего циклов.

Упражнение 8. Проанализируйте текст программы ввода натуральных чисел и печати на экране тех из них, которые являются совершенными. (Совершенное число — это число, равное сумме всех своих делителей, исключая само это число. Например, число 6 — совершенное, так как оно равно сумме всех своих делителей: $1 + 2 + 3$.) Ввод числа ноль означает завершение ввода.

```
program Demo_Sover;           {Определение совершенных чисел}
var
  S, N, I, J : integer;
```

```
begin
  repeat                       {Начало внешнего цикла}
  ite('Введите значение натурального числа >');
  N:=N;                         {Обнуление значения суммы}
  I:=1 to N-1 do {Внутренний цикл: поиск и суммирование делителей}
  if N mod I = 0 then S:=S+I; {Если число N без остатка делится на I, то
число I — делитель числа N, поэтому увеличить значение S на величину значения I}
  N:=S then WriteLn('Совершенное число ', N:6)
  else WriteLn('Несовершенное число ', N:6);
  until N=0;                     {Условие окончания внешнего цикла}
end.
```

Загрузите интегрированную среду программирования, введите текст программы `Demo_Sover`, откомпилируйте и исполните ее в режиме пошагового выполнения. Наблюдая за значениями параметров циклов I и S и выражений $I \leq N-1$ и $N=0$, проанализируйте действие операторов внешнего и внутреннего циклов. Контрольные данные: для N от 1 до 1000 совершенными являются числа 6, 28 и 496.

Правила пунктуации при записи операторов

При записи операторов необходимо соблюдать следующие правила пунктуации.

1. Точка с запятой не ставится в разделах описаний после зарезервированных слов `unit`, `uses`, `label`, `type`, `const`, `var` и ставится после завершения каждого описания.
2. Точка с запятой не ставится после `begin` и перед `end`, так как эти слова являются операторными скобками, а не операторами.
3. Точка с запятой является разграничителем операторов, ее отсутствие между операторами вызывает ошибку компиляции.
4. В операторах цикла точка с запятой не ставится после `while`, `repeat`, `do` и перед `until`.
5. В условных операторах точка с запятой не ставится после `then` и перед `else`.

Получение подсказки по языку программирования

Часто программисты, особенно начинающие, нуждаются в уточнении синтаксиса написания той или иной конструкции языка. Как уже было сказано в главе 1, интегрированная среда программирования позволяет оперативно получать подсказку по языку программирования. Для этого курсор устанавливается

на пустом зарезервированном слове и нажимается Ctrl+F1. Например, для оператора if по синтаксису оператора if следует установить курсор на if и нажать Ctrl+F1. На экране появится окно подсказки. Нажимая клавиши со стрелкой вниз, или PageDown, можно просмотреть весь текст подсказки. Пример из подсказки можно скопировать в окно редактирования. Для этого следует выбрать опцию Index пункта Help главного меню интегрированной среды программирования или нажать клавиши Shift+F1.

После нажатия Shift+F1 на экране появится окно со списком разделов помощи. С помощью клавиши со стрелкой вниз или PageDown следует переместить курсор на слово if и нажать Enter. Для копирования нужного фрагмента следует установить курсор в начало копируемого блока и мышью или клавишами со стрелками при нажатой клавише Shift выделить другим цветом, как показано на рис. 4.1.



Рис. 4.1. Окно подсказки для оператора if с помеченным блоком

Для получения дополнительных возможностей можно в окне подсказки нажать Alt+F10 для вызова локального меню, как показано на рис. 4.2.

Скопируйте блок в буфер обмена, для чего воспользуйтесь опцией Copy локального меню или нажмите Ctrl+Ins, а затем нажмите Esc для отказа от подсказки. После этого установите курсор в окне редактирования в позицию, куда нужно скопировать пример, и вызовите локальное меню.

Выберите в меню опцию Paste или нажмите Shift+Ins. Блок из буфера обмена будет скопирован в окно редактирования. Подробное описание использования справочной системы интегрированной среды программирования см. в приложении Б.

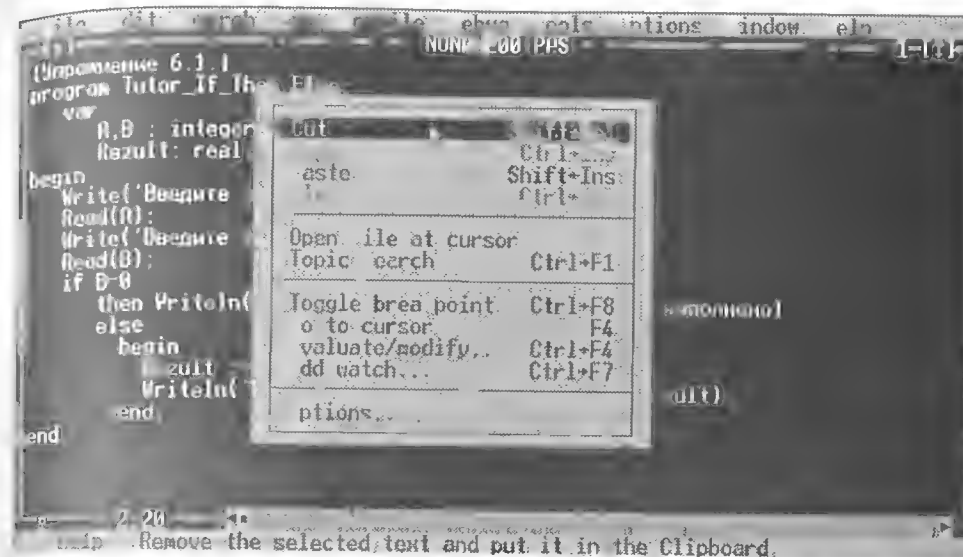


Рис. 4.2. Локальное меню окна редактирования

Тестирование и отладка программ

Как вы уже могли заметить, примеры и задания по мере изучения учебного материала усложняются. В связи с этим сложнее становится проверять правильность работы программ и обнаруживать ошибки в их текстах.

Для проверки правильности работы программы выполняется тестирование — исполнение программы с использованием некоторого набора входных данных, охватывающего весь спектр возможных значений для данного типа задач и проверяющего граничные условия, а также позволяющего посредством контроля промежуточных и конечных результатов решения задачи в ходе исполнения программы проверить выполнение операторов программы в требуемой последовательности и правильность действия всех алгоритмических конструкций (ветвлений, циклов, обращений к подпрограммам и т. п.).

Процесс тестирования удостоверяет качество программы, поэтому он должен быть документирован, т. е. будущие пользователи должны знать, как и при каких обстоятельствах программа тестировалась, каковы были входные данные и результаты, с тем чтобы тест можно было повторить.

Для обнаружения и устранения ошибок в программе выполняется ее отладка. Отладка в интегрированной среде программирования Turbo Pascal заключается в том, что с помощью встроенного в интегрированную среду специального средства — отладчика — анализируется поведение программы в «окрестностях» ошибки. С этой целью в интегрированной среде программирования обеспечивается возможность трассировки программы, т. е. выполнения «по

шагам» с остановкой в указанных точках или при выполнении заданных условий. Имеется возможность просмотра и изменения содержимого ячеек памяти и регистров процессора.

Эти возможности отладки доступны при выборе пункта Debug главного меню интегрированной среды программирования. Для просмотра значений переменных в процессе выполнения программы следует выбрать опцию Add watch, как показано на рис. 4.3.

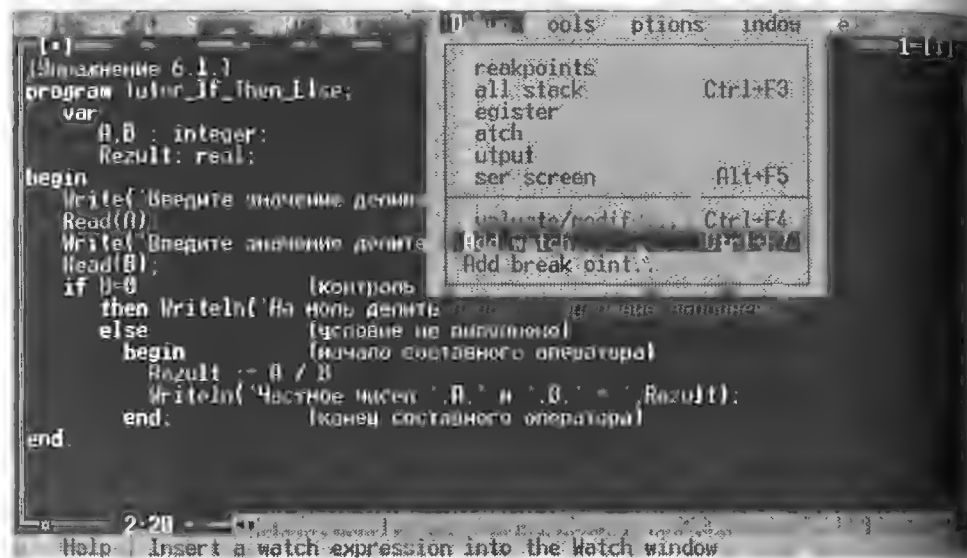


Рис. 4.3. Вид среды программирования с открытым меню Debug

Введите в окно Add watch выражение (например: $B = 0$) для наблюдения за его значением в ходе выполнения программы, как показано на рис. 4.4.

Для включения переменной или выражения в список просмотра можно, установив курсор на этой переменной или в начале выражения, нажать клавиши Ctrl+F7 и задать нужное выражение в появившемся окне (выражение копируется из окна редактирования). Для завершения следует нажать Enter или выбрать кнопку OK.

Для того чтобы в процессе отладки одновременно с окном редактирования на экране отображалось окно просмотра, следует выбрать в главном меню пункт Window и задать режим отображения окон Tile. После этого на экране будут одновременно отображаться и окно редактирования с текстом программы, и окно просмотра с текущими значениями выбранных для наблюдения переменных, как показано на рис. 4.5.

Если по ошибке был введен неверный идентификатор переменной или требуется удалить какой-либо идентификатор из окна просмотра, то следует нажать клавишу F6, выбрать нужный идентификатор и нажать клавишу Del.



Рис. 4.4. Окно ввода выражений для просмотра в процессе отладки

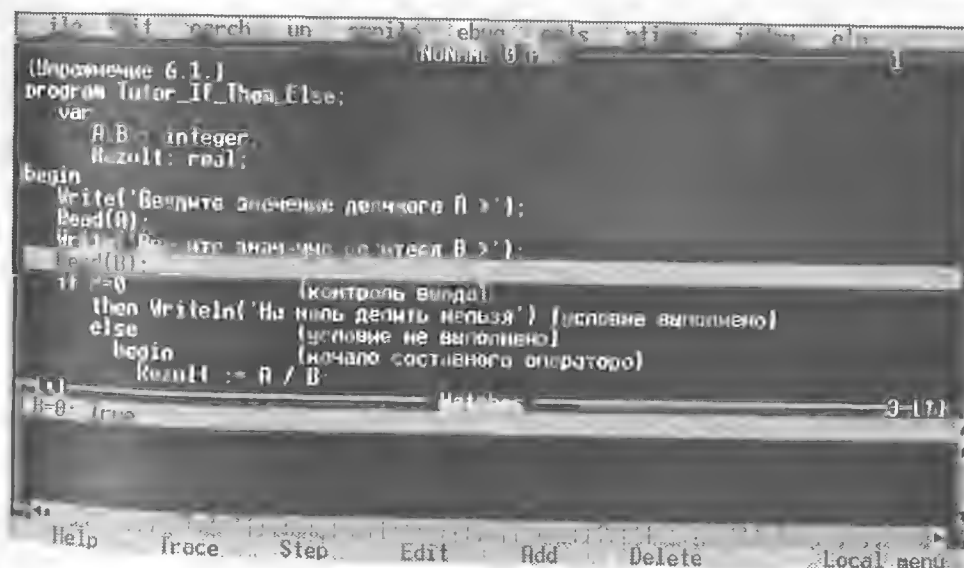


Рис. 4.5. Вид экрана с окнами редактирования и просмотра

Если требуется дополнить список переменных, значения которых просматриваются в процессе отладки, то следует нажать Ins и ввести идентификатор или соответствующее выражение.

Для отладки программы в режиме пошагового выполнения следует выбрать режим Step over меню Run главного меню интегрированной среды, как показано на рис. 4.6.

После запуска в этом режиме программа выполняется до строки, на которой расположен курсор, и останавливается, позволяя программисту просмотреть текущие значения переменных в данном месте программы. На рис. 4.9 показан просмотр текущих значений переменных в строке 14, где выполнение программы было прервано.

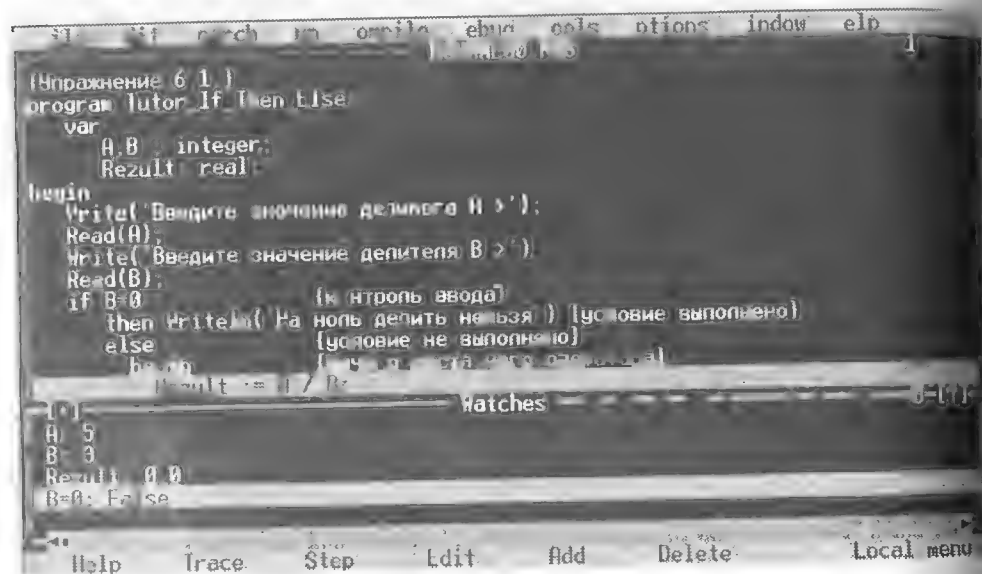


Рис. 4.9. Просмотр текущих значений переменных в строке 14, на которой расположен курсор

Чтобы закрыть окно просмотра, следует установить его в качестве текущего и нажать Alt+F3.

СОВЕТ

Работу достаточно сложных программ или их фрагментов, включающих сомнительные, на ваш взгляд, операторы, следует обязательно тестировать, используя режим отладки и просматривая текущие значения переменных.

Контрольные вопросы и задания

Вопросы

1. Что такое оператор? Чем отличаются простые и структурные операторы?
2. Опишите оператор присваивания, назначение и порядок выполнения.
3. Опишите оператор безусловного перехода, его назначение и особенности применения.
4. Объясните назначение оператора вызова процедуры.
5. В чем особенности пустого оператора? Его назначение?

6. Что представляет собой составной оператор? Как ограничиваются операторы, объединенные в составной оператор?
7. Опишите назначение, формы записи и порядок выполнения оператора условия if.
8. Опишите особенности использования вложенных условных операторов.
9. Зачем нужна отладка программ? Какие возможности для отладки программ предусмотрены в интегрированной среде программирования?
10. Каковы отличия оператора выбора case от оператора условия if?
11. Какие правила должны выполняться при использовании оператора выбора case?
12. Как оперативно получить подсказку по языку программирования в интегрированной среде программирования?
13. Каково назначение операторов повтора (цикла)?
14. Какие требования предъявляются к выражениям, управляющим повторениями?
15. В чем отличия операторов повтора while и repeat?
16. В каких случаях предпочтительнее использовать для организации циклов оператор повтора for? Что записывается в заголовке этого оператора?
17. Каким образом в операторе цикла for описывается направление изменения значения параметра цикла?
18. Какие ограничения накладываются на использование управляющей переменной (параметра цикла) в цикле for?
19. Какие правила пунктуации необходимо соблюдать при записи операторов?
20. Что такое вложенные циклы? Какие дополнительные условия необходимо соблюдать при организации вложенных циклов?

Задания

1. Создайте программу определения стоимости набора конфет, в который входят следующие сорта:

Наименование	Количество	Цена, р.	Наименование	Количество	Цена, р.
Красная шапочка	500 г	k	Воронежские	100 г	b
Алые паруса	200 г	A	Чародейка	250 г	v

2. Даны круг и квадрат. Создайте программу, определяющую по введенным значениям длин стороны квадрата и радиуса круга, верно ли утверждение «Круг вписан в квадрат». (Используйте логическую величину RESULT, принимающую значение TRUE, если утверждение истинно, и значение FALSE, если утверждение ложно.)

3. Создайте программу вычисления суммы цифр введенного с клавиатуры трехзначного натурального числа. Например: для числа 128 сумма цифр равна 11, для числа 345 сумма цифр равна 12.
4. Создайте программу, вычисляющую по введенному значению текущего времени (часов, минут, секунд) угол (в градусах) между положением часовой стрелки в начале суток и ее положением в текущее время. Например: если текущее время составляет 3 ч 30 мин 00 с, то этот угол составит $105^\circ 108'$.
5. Напишите программу-модель анализа пожарного датчика в помещении, которая выводит сообщение «Пожароопасная ситуация», если температура (в нашей модели она будет вводиться с клавиатуры) в комнате превысила 60°C .
6. Создайте программу, которая из двух введенных целых чисел печатает заключение о том, какое число больше.
7. Рис расфасован в два пакета. Вес первого — m кг, второго — n кг. Создайте программу, определяющую:
 - 1) какой пакет тяжелее — первый или второй?
 - 2) определите вес более тяжелого пакета.
8. Создайте программу, проверяющую, верно ли утверждение, что введенное целое число является четным.
9. Создайте программу, проверяющую, верно ли утверждение, что введенное целое число делится без остатка на 3.
10. Напишите программу, которая анализирует возраст человека и относит его к одной из четырех групп: дошкольник, ученик, рабочий, пенсионер. Возраст вводится с клавиатуры.
11. Создайте программу, определяющую, входит ли введенная цифра в десятичную запись введенного трехзначного числа, и печатающую сообщение о том, входит ли эта цифра в запись числа или нет.
12. Создайте программу, определяющую, лежит ли точка с указанными координатами X, Y на окружности радиуса R с центром в начале координат.
13. Создайте программу, определяющую, пройдет ли график функции $y = 3x^2 - 7x + 2$ через заданную точку с координатами (a, b) .
14. К финалу конкурса лучшего по профессии «Специалист электронного офиса» были допущены трое: Иванов, Петров, Сидоров. Соревнования проходили в три тура. Иванов в первом туре набрал m_1 баллов, во втором — n_1 , в третьем — p_1 . Петров — соответственно m_2, n_2, p_2 ; Сидоров m_3, n_3, p_3 . Создайте программу, определяющую, сколько баллов набрал победитель.
15. Создайте программу, которая по трем введенным числам определит, могут ли эти числа быть длинами сторон треугольника, и если да, то каков б...

тип полученного треугольника с данными длинами сторон (прямоугольный, остроугольный, тупоугольный).

16. Квадраты при игре в крестики-нолики перенумерованы, как показано на рисунке. Заданы номера трех квадратов: N_1, N_2, N_3 , причем $N_1 < N_2 < N_3$.

1	2	3
4	5	6
7	8	9

Проверьте, лежат ли квадраты:

- 1) на одной диагонали;
 - 2) на одной вертикали;
 - 3) на одной горизонтали.
17. Напишите программу-фильтр, которая при нажатии любых клавиш выводит на экран только буквы и цифры, при этом указывая, что именно выводится: буква или цифра.
 18. Напишите программу, которая по паролю будет определять степень доступа сотрудника к секретной информации в базе данных. Доступ к базе имеют только шесть человек, разбитых на три группы по степени доступа. Они имеют следующие пароли:
 - 9583, 1747 — доступны модули базы А, Б, В
 - 3331 7922 — доступны модули базы Б, В
 - 9455, 8997 — доступен модуль базы В
 19. Создайте программу, реализующую пример применения компьютера в книжном магазине. Компьютер запрашивает стоимость книг, сумму денег, внесенную покупателем; если сдачи не требуется, печатает на экране «спасибо»; если денег внесено больше, то печатает «возьмите сдачу»; выводит сумму сдачи; если денег недостаточно, то печатает соответствующее сообщение и указывает размер недостающей суммы.
 20. В компьютер поступают результаты соревнований по плаванию для трех спортсменов. Создайте программу, которая выбирает лучший результат и выводит его на экран с сообщением, что это результат победителя плавания.
 21. Создайте программу, которая по введенному k — числу грибов — печатает фразу «Мы нашли в лесу k грибов», причем согласовывает окончание слова «гриб» с числом k . (Количество грибов может быть любым целым числом: 1, 3, 34, 127 и т. п. Окончание фразы определяется последней цифрой.)

22. Создайте программу, которая для введенного целого числа k (от 1 до 99) напечатает фразу «Мне k лет», где k — введенное число, при этом в нулевых случаях слово «лет» заменяя на слово «год» или «года». (Например: при $k=70$ «Мне 70 лет», при $k=15$ «Мне 15 лет», при $k=23$ «Мне 23 года», и т. п.).
23. Создайте программу для вычисления числа дней в месяце, если даны: номер месяца N — целое число от 1 до 12, целое число A , равное 1 для високосного года и 0 в противном случае.
24. Создайте программу, которая вычисляет сумму чисел от 1 до N . Значение N (N должно быть меньше 100) вводится с клавиатуры.
25. Напишите программу печати таблицы перевода расстояний из дюймов в сантиметры ($1 \text{ дюйм} = 2,5 \text{ см}$) для значений длины от 1 до 20 дюймов.
26. Используя цикл `while`, напишите программу вывода всех четных чисел с 2 до 100 включительно.
27. Создайте и произведите отладку программы, вычисляющей сумму квадратов чисел от 1 до введенного целого числа n .
28. Используя цикл `while`, напишите программу определения суммы всех нечетных чисел от 1 до 99 включительно.
29. Используя цикл `while`, напишите программу определения идеального веса для взрослых людей по формуле: $\text{Ид. вес} = \text{рост} - 100$. Выход из цикла значение роста = 250.
30. Используя цикл `repeat`, напишите программу-фильтр, которая вводит любые символы, но комментирует только буквы русского алфавита. Завершение работы программы — по нажатии буквы «Я».
31. Используя цикл `repeat`, напишите программу, которая требует ввод пароля, например, числа 111, и если пароль правильный, то заполняет все строки экрана сообщением «Молодец!!!». Если после пятой попытки пароль неверен, выйти из программы.
32. Создайте программу получения в порядке убывания всех делителей данного числа.
33. Создайте программу определения наибольшего общего делителя двух натуральных чисел.
34. Создайте программу определения наименьшего общего кратного двух натуральных чисел.
35. Создайте программу, подсчитывающую количество цифр введенного целого неотрицательного числа. (Можно использовать операцию целочисленного деления для последовательного уменьшения числа на один разряд.)
36. Создайте и произведите отладку программы, определяющей максимальное из всех введенных чисел. (Пусть признаком конца ввода чисел служит число 0.)

37. Найдите наибольшее и наименьшее значение функции $y = 3x^2 + x - 4$, если в заданном интервале $[a, b]$ x изменяется с шагом 0,1.
38. Вычислите сумму квадратов N четных натуральных чисел.
39. Вычислите:
 - 1) $1 + 2 + 4 + 8 + \dots + 2^{10}$
 - 2) $(1 + 2) \cdot (1 + 2 + 3) \cdot \dots \cdot (1 + 2 + \dots + 10)$
40. В бригаде, работающей на уборке сена, имеется N косилок. Первая из них работала m часов, а каждая следующая на 10 минут больше, чем предыдущая. Сколько часов проработала вся бригада?

Глава 5

Процедуры и функции

Методы программирования

Необходимость структуризации в программировании

Значительное увеличение сложности задач, решаемых с помощью компьютеров, приводит к увеличению размеров и сложности программ, что порождает трудности при их разработке и отладке. Увеличение продолжительности жизненного цикла программ приводит к тому, что с течением времени из-за изменения условий использования программ возникает необходимость их модификации, повышения их эффективности, удобства пользования ими.

Для решения возникающих при этом проблем в практике программирования выработан ряд приемов и методов, которые принято называть методами структурного программирования. Под *структурным программированием* понимают такие методы разработки и записи программы, которые ориентированы на максимальное удобство восприятия и понимания ее человеком. При прочтении программы в ее последовательных фрагментах должна четко прослеживаться логика работы, т. е. не должно быть «скачков» на фрагмент программы, расположенные где-то в другом месте программы.

Структурное программирование является «программированием без *go to*», т. е. без крайней необходимости не используются операторы перехода. В связи с этим отдельные фрагменты программы представляют собой некоторые логические (управляющие) структуры, которые определяют порядок выполнения содержащихся в них правил обработки данных. Любая программа получается построенной из стандартных логических структур, число типов которых невелико.

Перечислим основные логические структуры:

- *следование* — последовательность операторов или групп операторов, выполняемых друг за другом в порядке их следования в тексте программы;
- *ветвление* — управляющая структура, которая в зависимости от выполнения заданного условия определяет выбор для выполнения одного из или более заданных в этой структуре групп операторов;

- *повторение* — цикл, в котором группа операторов может выполняться повторно, если соблюдается заданное условие.

Существенная особенность всех этих структур — то, что каждая из них имеет только один вход и только один выход, что и обеспечивает логически последовательную структуру программы. Все эти структуры определяются рекурсивно, т. е. каждая из входящих в структуру групп операторов может быть одним оператором или группой операторов и может представлять собой любую из допустимых структур — допускается вложение структур.

Так как программа задает правила обработки данных, то проектирование самих данных при создании программы имеет не менее важное значение, чем проектирование правил их обработки. Очевидно, что чем четче определены сами данные, тем легче разрабатывать правила их обработки. Простота и надежность программы существенно зависят от того, насколько удобно отдельные обрабатываемые данные объединены в структуры. При этом язык программирования Pascal требует от программиста четкого описания вводимой структуры данных, что позволяет транслятору обеспечивать работу с каждой такой структурой и следить за корректностью ее использования.

Чтобы возложить на транслятор контроль за корректностью использования в программе различных типов данных, в Pascal требуется описывать константы и переменные перед их применением с указанием их типа. Чтобы эти описания было легче использовать транслятору или программисту, Pascal требует четкой структуризации программы, отводя для каждого из типов информации строго определенное место (см. структуру программы).

Метод нисходящего проектирования программ

С массовым внедрением вычислительной техники процесс программирования постепенно превращается в промышленное изготовление программ. Для этой цели создаются разнообразные технологии программирования. Примерами таких технологий могут служить технология нисходящего программирования и технология восходящего программирования.

Технология нисходящего программирования базируется на методе программирования «сверху-вниз». Часто этот метод называют *методом пошаговой детализации*. Большинство специалистов в области программирования придерживаются той точки зрения, что именно этот метод создает предпосылки к решению сложных проблем. Основой этого метода является идея постепенной декомпозиции исходной задачи на ряд подзадач. Сначала формулируется самая грубая модель решения, отдельные детали которой на первом этапе могут быть довольно расплывчатыми, как, например, вид участка земли с большой высоты, при котором неразличимы мелкие подробности. По мере разработки программы, разбивая наиболее неясные части алгоритма и добиваясь все более точных и детализированных формулировок, мы получаем более подробное решение, как бы опускаемся с большой высоты ниже и начинаем

при этом различать более мелкие детали. Решение отдельного фрагмента сложной задачи может представлять собой самостоятельный программный блок, называемый *подпрограммой*. Такой процесс детализации продолжается до тех пор, пока не станут ясны все детали решения задачи. В этом случае программному решению сложной задачи можно представить в виде иерархической совокупности сравнительно самостоятельных фрагментов — подпрограмм.

Таким образом, подпрограммой называют обособленную, оформленную в виде отдельной синтаксической конструкции и снабженную именем часть программы. Использование подпрограмм позволяет, сосредоточив в них подробное описание некоторых операций, в остальной программе только указывать имена подпрограмм, чтобы выполнить эти операции. Возможны неоднократные вызовы подпрограмм из разных участков программы, причем при вызове подпрограмме можно передать некоторую информацию (различную для разных вызовов), чтобы одна и та же подпрограмма выполняла решение подзадачи для разных случаев.

Понятие подпрограммы как обособленной именованной части программы со своими собственными объектами (константами, переменными и т. п.) является во многих языках программирования основным средством структурирования программ. Современные подходы к разработке программ поощряют явное оформление в виде подпрограммы любого достаточно самостоятельно и законченного программного фрагмента.

Подпрограммы в языке Pascal

За наличие подпрограмм как средства структурирования программ язык программирования Turbo Pascal называется *процедурно-ориентированным*. Подпрограммы в Turbo Pascal реализованы посредством процедур и функций. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

Процедура — это независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Структура процедуры повторяет структуру программы. Процедура не может выступать в качестве операнда в выражении. Упоминание имени процедуры в тексте программы приводит к активизации процедуры и называется ее *вызовом*. Например, `Read(F)` читает с клавиатуры некоторое значение и присваивает его переменной `F`, `Delay(5)` вызывает задержку выполнения программы на 5 мс.

Функция аналогична процедуре, но имеются два отличия: функция передает в точку вызова скалярное значение; имя функции может входить в выражение в качестве операнда. Например, функция `Chr(65)` возвращает в точку вызова символ `A` (код ASCII — 65), `Sqr(X)` — возводит в квадрат значения целого или вещественного `X` и возвращает в точку вызова вычисленное значение квадрата числа `X`.

Итак, отличие подпрограмм-процедур от подпрограмм-функций состоит в том, что процедуры служат для задания совокупности действий, направленных на изменение внешней по отношению к ним программной обстановки, а функции, являясь частным случаем процедур, отличаются от них тем, что они обязательно возвращают в точку вызова основной программы единственный результат как значение имени этой функции.

Все процедуры и функции языка Turbo Pascal делятся на две группы: встроенные (стандартные) и определенные пользователем. Первые входят в состав языка и вызываются для выполнения по строго фиксированному имени. Вторые разрабатываются и именуется самим пользователем. Все стандартные средства располагаются в специализированных библиотечных модулях, которые имеют системные имена.

Стандартные библиотечные модули

В систему Turbo Pascal версии 6.0 и старше входят восемь модулей: `System`, `Crt`, `Dos`, `Graph`, `Graph3`, `Overlay`, `Printer`, `Turbo3` и специализированная библиотека `Turbo Vision`. Модуль `System` подключается по умолчанию, все остальные должен подключать программист с помощью зарезервированного слова `uses`. Например: `uses Crt, Dos, Printer`.

Рассмотрим кратко назначение каждого из модулей.

`System` — «сердце» Turbo Pascal; содержащиеся в нем подпрограммы обеспечивают работу всех остальных модулей системы.

`Crt` — содержит средства управления дисплеем и клавиатурой компьютера.

`Dos` — включает средства, позволяющие реализовывать различные функции DOS.

`Graph3` — поддерживает использование стандартных графических подпрограмм версии Turbo Pascal 3.0.

`Overlay` — содержит средства организации оверлейных программ.

`Printer` — обеспечивает быстрый доступ к печатающему устройству.

`Turbo3` — обеспечивает максимально возможную совместимость с версией Turbo Pascal 3.0.

`Graph` — содержит пакет графических средств, обеспечивающих эффективную работу с адаптерами CGA, EGA, VGA, HERC, IBM 3270, MCGA и ATT6300.

`Turbo Vision` — библиотека объектно-ориентированных подпрограмм для разработки пользовательских интерфейсов.

Встроенные функции и процедуры

Модуль `System` подключается к программе автоматически, поэтому его имя не указывается в разделе `uses`. По этой причине программе становятся доступны его встроенные процедуры и функции.

Арифметические процедуры и функции

Abs(X:real/integer):real/integer — вычисление абсолютной величины X . Тип результата совпадает с типом параметра.

Arctan(X:real):real — вычисление угла, тангенс которого равен X радиан.

Cos(X:real):real — вычисление косинуса X ; параметр задает значение угла в радианах.

Exp(X:real):real — вычисление экспоненты X , т. е. значение числа e в степени X , число e является основанием натурального логарифма и равно 2.718282.

Frac(X:real):real — вычисление дробной части X .

Int(X:real):real — вычисление целой части X .

Ln(X:real):real — вычисление натурального логарифма X , т. е. логарифма по основанию e ($e = 2.718282$).

Pi:real — возвращает значение числа π (3.141592653897932385).

Sin(X:real):real — вычисление синуса X . Параметр задает значение угла в радианах.

Sqr(X) — возведение в квадрат значения целого или вещественного значения X . Тип результата совпадает с типом параметра.

Sqrt(X:real):real — вычисление квадратного корня из X .

Random:real — генерирует значение случайного числа из диапазона 0..0.99.

Random(I:word):word — генерирует значение случайного числа из диапазона 0..I.

Randomize — изменение базы генератора случайных чисел.

Скалярные процедуры и функции

Dec(X{n}) — процедура уменьшает значение целочисленной переменной X на величину n . При отсутствии необязательного параметра n значение X уменьшается на единицу.

Inc(X{n}) — процедура увеличивает значение целочисленной переменной X на n . При отсутствии необязательного параметра n значение X увеличивается на единицу.

Pred(S) — функция возвращает элемент, предшествующий S в списке значений типа. Тип результата совпадает с типом параметра. Если предшествующего S элемента не существует, возникает программное прерывание.

Succ(S) — функция возвращает значение, следующее за S в списке значений типа. Тип результата совпадает с типом параметра. Если следующее за S значение отсутствует, возникает программное прерывание.

Odd(I:integer):boolean — возвращает True, если I нечетное, и False, если I четное.

Функции преобразования типов

Chr(I:byte):char — возвращает символ кода ASCII с номером, равным значению I . Если значение параметра больше 255, возникает программное прерывание.

Ord(S):longint — возвращает порядковый номер значения S в множестве, определенном типом S .

Round(X:real):longint — возвращает значение X , округленное до ближайшего целого числа.

Trunc(X:real):longint — возвращает ближайшее целое число, меньшее или равное X , если $X \geq 0$, и большее или равное X , если $X < 0$.

Процедуры управления программой

Delay(I:word) — задержка выполнения программы на I мс.

Exit — выход из выполняемого блока в окружающую среду. Если текущий блок является процедурой или функцией, выход производится во внешний блок. Если **Exit** указана в операторной части основной программы, программа прекращает работу, и управление передается системе программирования.

Halt(N:word) — прекращение выполнения программы и передача управления системе программирования (если выполнялся .PAS-файл) или DOS (если выполнялся .EXE-файл). N — код завершения программы, передаваемый в операционную систему.

RunError(ErrCode:word) — прекращение выполнения программы и генерация ошибки этапа выполнения. **ErrCode** — параметр типа **byte**, содержащий номер ошибки.

Специальные процедуры и функции

FillChar(P,D,Z) — заполняет побайтно область основной памяти заданным значением (заполнителем). Является одной из наиболее быстродействующих процедур. Область начинается с первого байта указанной переменной P и имеет размер, заданный параметром D . P — переменная любого типа; D — целочисленное выражение, указывающее длину; Z — заполнитель, выражение литерного или байтового типа.

Move(P1,P2,D) — пересылает содержимое основной памяти, начиная с первого байта переменной $P1$, в область, которая начинается с первого байта переменной $P2$. Длина областей определяется параметром D . $P1$ и $P2$ — переменные любого типа; D — целочисленное выражение.

Hi(I:integer):byte — выделяет старший байт значения I и помещает его в младший байт результата. Старший байт результата равен 0.

Lo(I:integer):byte — выделяет младший байт значения I и помещает его в младший байт результата. Старший байт результата равен 0.

ParamCount: string — возвращает число параметров, переданных программе в командной строке.

ParamStr(n:word): string — возвращает указанный параметр командной строки.

SizeOf(IT):word — вычисляет объем основной памяти в байтах, которую занимает указанная переменная или тип. IT — идентификатор переменной или типа данных.

Swap(I:integer):integer — меняет местами содержимое младшего и старшего байтов целочисленного выражения, заданного параметром I типа integer.

Вызов стандартной процедуры или функции

Ранее мы уже рассматривали примеры программ, в которых использовались некоторые стандартные процедуры и функции. Для использования стандартной процедуры или функции к программе подключается тот или иной специализированный библиотечный модуль, в который входит данная стандартная процедура или функция (исключение составляет модуль System, так как он подключается к программе автоматически), для чего имя специализированного библиотечного модуля указывается в разделе uses. Затем в программе осуществляется вызов процедуры или функции, для чего записывается ее имя и указываются фактические параметры, например: Pi, Sin(X), Chr(125), Inc(X5). Так как после выполнения функции ее значение присваивается имени, то имя функции используется в выражении.

Процедуры и функции пользователя

Если в программе возникает необходимость частого обращения к некоторой группе операторов, то рационально сгруппировать такую группу операторов в самостоятельный блок, к которому можно обращаться, указывая его имя. Такие разработанные программистом самостоятельные программные блоки называются *подпрограммами пользователя*. Они являются основой модульного программирования. Разбивая задачу на части и оформляя логически обособленные модули в виде процедур и функций, программист реализует основные принципы широко используемого в практике системного подхода и методов нисходящего программирования.

При вызове подпрограммы (процедуры или функции), определенной программистом, работа главной программы на некоторое время приостанавливается и начинает выполняться вызванная подпрограмма. Она обрабатывает данные, переданные ей из главной программы. По завершении выполнения подпрограмма-функция возвращает главной программе результат (подпрограмма-процедура не возвращает явно результирующего значения).

Передача данных из главной программы в подпрограмму и возврат результата выполнения функции осуществляются с помощью параметров. *Параметром* называется переменная, которой присваивается некоторое значение в рамках указанного применения. Различают *формальные параметры* — параметры, определенные в заголовке подпрограммы, и *фактические параметры* — выражения, задающие конкретные значения при обращении к подпрограмме. При обращении к подпрограмме ее формальные параметры замещаются фактическими, переданными из главной программы. Механизм передачи параметров будет рассмотрен позднее.

Процедуры

Описание процедуры включает заголовок (имя) и тело процедуры. Заголовок состоит из зарезервированного слова procedure, идентификатора (имени) процедуры и необязательного, заключенного в круглые скобки, списка формальных параметров с указанием типа каждого параметра. Имя процедуры — идентификатор, уникальный в пределах программы. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе. Общая структура описания процедур и функций иллюстрируется следующими синтаксическими диаграммами.

Описание процедуры:



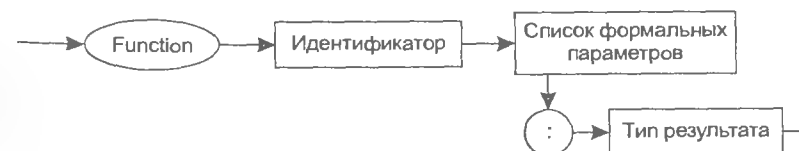
Описание функции:



Заголовок процедуры:



Заголовок функции:



Описания меток, констант, типов и т. д. действительны только в пределах данной процедуры. В теле процедуры можно использовать любые глобальные константы и переменные.

```

procedure <имя> (Формальные параметры);
const
type
var
begin
  <операторы>
end;
  
```

В качестве примера опишем процедуру, которая прерывает выполнение программы и выдает соответствующее сообщение об ошибке:

```
procedure Abort(Msg: string);
begin
  WriteLn('Ошибка: ', Msg);
  Halt(1);
end;
```

В данной пользовательской процедуре использована переменная *Msg* типа *string*, в которой хранится текст сообщения о характере ошибки, вызвавшей прерывание программы. Для прерывания выполнения программы используется стандартная процедура *Halt* из стандартного библиотечного модуля *System*.

Процедура не может выполняться сама, ее необходимо вызвать по имени и указать фактические параметры того же типа, что и формальные. Количество и тип формальных параметров строго соответствуют количеству и типу фактических параметров. В качестве примера приведем фрагмент программы, в котором используется описанная выше процедура *Abort*:

```
program DemoProc; {Подсчет суммы десяти введенных целых положительных чисел: если
будет введено отрицательное число, прервать выполнение}
const
  Limit = 10; {Ограничение на количество вводимых чисел}
var
  Count, Item, Sum : integer;
{$I ABORT.PAS} {Директива компилятору на включение текста программы ABORT.PAS
в качестве подпрограммы}
begin
  Count:= 0;
  Sum:= 0;
  while (Count < Limit) do {Условие выполнения цикла}
  begin
    Count:= Count+1;
    Write('Введите ' Count, '-е целое число: ');
    ReadLn(Item);
    if Item < 0 then {Если введено отрицательное число}
      Abort('введено отрицательное число'); {Вызов процедуры}
    Sum:= Sum+Item;
  end;
  WriteLn('Сумма введенных чисел равна ' Sum);
end;
```

В разделе описания программы описывается константа *Limit*, ограничивающая количество вводимых чисел; в разделе описания переменных описываются переменные *Count*, *Item*, *Sum* типа *integer*. Затем в блоке описания записана директива компилятора *{\$I ABORT.PAS}*, указывающая, что при компиляции данной программы в нее нужно включить в качестве процедуры программу *ABORT.PAS*.

В начале программы обнуляются значения количества введенных чисел *Count* и их сумма *Sum*. Потом выполняется цикл, пока очередное вводимое число меньше предельного, заданного значением константы *Limit*. Сначала устанавливается номер очередного числа, затем на экран выводится приглашение «Введите 1-е (2-е и т. п.) число», и считывается значение числа с клавиатуры в переменную *Item*. Затем проверяется условие *Item < 0*.

Если условие выполняется, то вызывается внешняя процедура *Abort*, которой передается фактический параметр-значение «введено отрицательное число». Это значение присваивается формальному параметру *Msg* процедуры *Abort*. Процедура *Abort* выводит на экран сообщение «Ошибка» и печатает текст сообщения — значение параметра *Msg* «введено отрицательное число», после чего вызывает стандартную процедуру *Halt(1)*, которая прерывает выполнение программы.

Если условие *Item < 0* не выполняется, то значение суммы *Sum* увеличивается на значение введенного числа *Item*, и управление передается в заголовок цикла для проверки условия *Count < Limit*. Если условие соблюдается, то тело цикла выполняется еще раз, иначе цикл завершается, а управление в программе передается на оператор, следующий за циклом, т. е. за зарезервированным словом *end*; обозначающим окончание составного оператора в теле цикла. После этого на экран выводится сообщение «Сумма введенных чисел равна» и печатается значение переменной *Sum*. На этом выполнение программы завершается.

Упражнение 1. Запустите интегрированную среду программирования. Введите в первое окно редактора текст программы *DemoProc*, а во второе окно — текст процедуры *Abort* и запишите файлы на диск под соответствующими именами.

ПРИМЕЧАНИЕ

Чтобы открыть новое окно, выберите опцию *New* меню *File* главного меню интегрированной среды программирования. Для перехода из одного окна в другое используется клавиша *F6*.

Откомпилируйте файл *DemoProc*. Если появятся сообщения об ошибках, внесите исправления и откомпилируйте программу снова. После того как компиляция выполнится успешно, задайте для просмотра в окне отладчика величины *Sum*, *Item* и выражение *Count < Limit*. Для того чтобы одновременно на экране отображались окна с текстами программы и процедуры, а также окно просмотра значений переменных, в пункте *Window* главного меню интегрированной среды следует задать опцию *Tile*. Нажатием клавиши *F7* запустите программу на выполнение в пошаговом режиме с трассировкой процедур. Экран будет выглядеть, как показано на рис. 5.1.

Нажимая клавишу *F7*, наблюдайте за изменением значений переменных и выражений в окне просмотра.

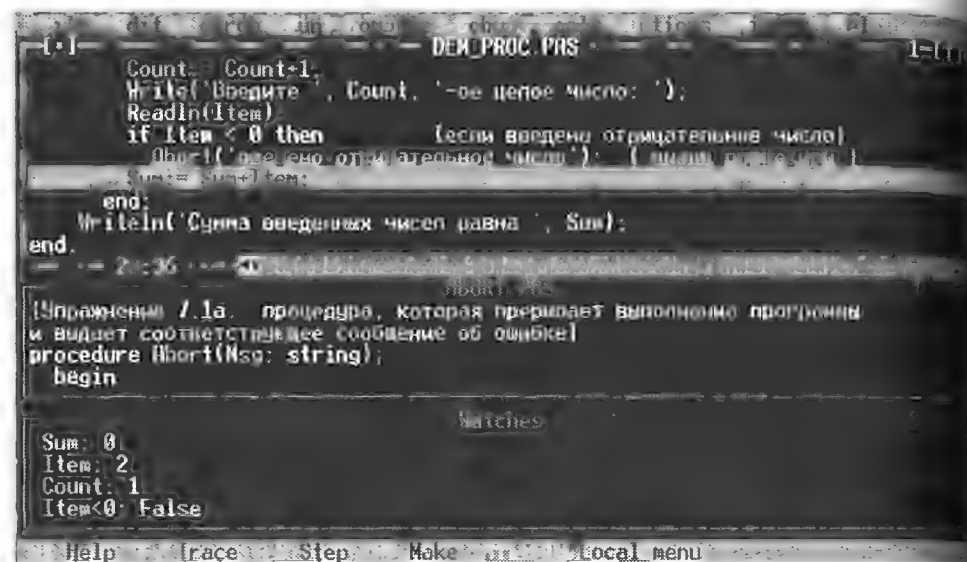


Рис. 5.1. Вид экрана с окнами программы, процедуры и просмотра

В ответ на запрос «Введите число» введите отрицательное число и обратите внимание на соблюдение условия $Item < 0$, вследствие чего из основной программы вызывается процедура `Abort`. При этом в формальный параметр процедуры — переменную `Msg` — передается значение сообщения «Введено отрицательное число». Результатом действия процедуры будет вывод на экран сообщения «Ошибка: введено отрицательное число» и прерывание работы программы.

Как видно из примера, параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными строковыми сообщениями.

Если процедура возвращает в программу какие-либо значения, соответствующие переменные должны быть описаны как параметры-переменные с использованием слова `var` (см. Параметры).

Функции

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово `function`, идентификатор (имя) функции, заключенный в круглые скобки, необязательный список формальных параметров и тип возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре аналогичный программе:

```
function <имя> (Формальные параметры) : <тип результата>;
const ...;
type ...;
var ...;
begin
  <операторы>
end;
```

В разделе операторов должен присутствовать, по крайней мере, один оператор, присваивающий имени функции значение. В точку вызова возвращается результат последнего такого присваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип. В качестве примера приведем программу вычисления выражения $Z = (A^5 + A^{-3}) / 2 \cdot A^M$, в которой возведение в степень выполняется функцией `Step`.

{Программа вычисления выражения $Z = (A^5 + A^{-3}) / 2 \cdot A^M$ }

program DemoFunc;

var

M : integer;

A, Z, R : real;

{функция вычисления степени N, X — формальные параметры, результат, возвращаемый функцией в точку вызова, имеет вещественный тип}

function Step(N : integer; X : real) : real;

var

I : integer;

Y : real;

begin

Y:=1;

for I:=1 to N do {Цикл вычисления N-й степени числа X}

Y:=Y*X;

Step:=Y {Присваивание функции результата вычисления степени}

end;

{Конец процедуры-функции}

begin

{Начало основной программы}

Write('Введите значение числа A и показатель степени M');

Readln(A,M);

Z:=Step(5,A); {Вызов функции с передачей ей фактических параметров 5, A}

Z:=Z+ Step(3,1/A); {Вызов функции с передачей ей фактических параметров 3, 1/A}

if M=0 then R:=1

else if M>0 then R:=Step(M,A) {Вызов функции с передачей ей фактических параметров M, A}

else R:=Step(-M,A); {Вызов функции с передачей ей фактических параметров -M, A}

Z:=Z/(2*R);

Writeln('Для A=' . A . ' M=' . M . ' Значение выражения =' . Z);

end.

В начале программы описывается переменная целого типа `M` и переменные вещественного типа `A`, `Z`, `R`, после этого описывается функция `Step` вычисления степени числа, с формальными параметрами `N` и `X`, результат, возвращаемый функцией в точку вызова, — вещественного типа.

В описании функции вводятся две локальные (местные) переменные `I` и `Y`. Переменная `I` служит для подсчета числа повторений цикла, а в `Y` накапливается

значение степени как произведения N одинаковых сомножителей. В конце функции `Step` присваивается значение вычисленного произведения.

В начале выполнения основной программы на экран выводится запрос «Введите значение числа A и показатель степени M », и считывается с клавиатуры значение вещественного числа A и целого числа M .

Затем выполняется оператор `Z:=Step(5,A)`. Сначала осуществляется вызов функции `Step` с передачей ей фактических параметров 5 , A . Их значения присваиваются формальным параметрам функции N и X . По окончании вычисления степени числа значение функции `Step`, вычисленное для фактических параметров 5 и A , присваивается переменной Z . Аналогично в операторе `Z:=Z+Step(3,1/A)` сначала осуществляется вызов функции `Step` с передачей ей фактических параметров 3 , $1/A$, после чего значение переменной Z увеличивается на величину возвращенного в основную программу результата вычисления функции `Step`.

Оператор `if M=0 then R:=1 else if M>0 then R:=Step(M,A) else R:=Step(-M,A)` проверяет условия $M=0$, $M>0$ и, в зависимости от их соблюдения, либо присваивает переменной R значение 1 (при $M=0$), либо выполняет вызов функции `Step` для фактических значений M , A или $-M$, A , а после вычисления значения функции `Step` присваивает его переменной R . Оператор `Z:=Z/(2*R)` выполняет вычисление значения выражения $Z/(2*R)$, а затем присваивает вычисленное значение переменной Z .

В конце программы стандартная процедура `Writeln` выводит на экран сообщение о результате вычисления степени M числа A .

Упражнение 2. Запустите интегрированную среду программирования. Введите текст программы `DemoFunc` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции задайте для просмотра в окне отладчика величины A , M , I , Y , `Step`, Z , R . Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Выполните программу в пошаговом режиме с трассировкой функций и проследите за изменениями значений переменных в основной программе и в подпрограмме-функции, обратите внимание на передачу значений при вызове функции от фактических параметров основной программы формальным параметрам функции и возврат вычисленного функцией значения в основную программу.

Механизм передачи параметров

Как было показано в приведенных выше примерах программ с использованием процедур и функций, в заголовке процедуры или функции может быть задан список параметров, которые называются формальными. Название «формальные» эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут подставлены конкрет-

ные значения. Этот список указывается после имени подпрограммы и заключается в круглые скобки.

Список формальных параметров, указываемых в заголовке подпрограммы, может включать в себя:

- параметры-значения;
- параметры-переменные, перед которыми должно стоять служебное слово `var` и за которыми указывается их тип;
- параметры-процедуры, перед которыми должно стоять служебное слово `procedure`;
- параметры-функции, перед которыми должно стоять служебное слово `function` и после которых указывается тип значения, возвращаемого функцией в основную программу;
- нетипизированные параметры, перед которыми должно стоять служебное слово `var`, а указание типа должно отсутствовать.

В списке должны быть перечислены имена формальных параметров и их типы. Имя параметра отделяется от типа двоеточием, а параметры друг от друга — точкой с запятой. Имена параметров одного типа можно объединять в подписки, в которых имена отделяются друг от друга запятой.

Примеры заголовков:

```
procedure P(procedure B; function C : real; Q, W, R : char);
procedure A;
```

Между формальными и фактическими параметрами должно быть полное соответствие:

- формальных и фактических параметров должно быть одинаковое количество;
- порядок следования фактических и формальных параметров должен совпадать;
- тип каждого фактического параметра должен совпадать с типом соответствующего формального параметра.

Параметры-значения

Параметры-значения используются только для передачи исходных данных из основной программы в подпрограмму (процедуру или функцию). В списке формальных параметров они перечисляются через запятую с обязательным указанием их типов, как было, например, в приведенных выше примерах:

```
procedure Abort(Msg: string);
function Step(N : integer; X : real): real;
```

Если формальный параметр объявлен как параметр-значение, то фактическим параметром может быть произвольное выражение. При вызове подпрограммы фактические параметры вычисляются и используются в качестве началь-

ных значений формальных параметров, т. е. осуществляется подстановка значений. Если формальный параметр определен как параметр-значение, то перед вызовом процедуры это значение вычисляется, полученный результат помещается во временную память и передается процедуре. Даже если фактический параметр — простейшее выражение в виде константы или переменной, все равно процедуре будет передана лишь копия этой константы (переменной). В процессе выполнения подпрограммы формальные параметры могут изменяться, но это никак не отражается на соответствующих фактических параметрах-переменных, которые сохраняют те значения, которые имели до вызова подпрограммы, так как меняются не они, а их копии. Поэтому параметры-значения нельзя использовать для передачи результатов из подпрограммы в основную программу.

Пример программы с использованием передачи параметров по значению:

```
program Par_Znach;
var
  A, B : real; {Процедура вычисления квадратов двух чисел и вывода на экран их
  суммы}
  procedure Sum_Square(X, Y : real); {X, Y — формальные параметры}
  begin
    X := X * X;
    Y := Y * Y;
    WriteLn('Сумма квадратов =', X + Y);
  end;
begin
  {Конец процедуры}
  {Начало основной программы}
  A := 1.5;
  B := 3.4;
  Sum_Square(A, B); {Вызов процедуры Sum_Square с передачей ей значений фактических
  параметров A и B}
end.
```

При вызове процедуры Sum_Square с фактическими параметрами A, B значения этих параметров однократно копируются в соответствующие формальные параметры X, Y, и дальнейшие преобразования формальных параметров X, Y внутри процедуры Sum_Square уже никак не влияют на значения переменных A и B.

Упражнение 3. Запустите интегрированную среду программирования. Введите текст программы Par_Znach, запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После того как компиляция выполнится успешно, задайте для просмотра в окне отладчика величины A, B, X, Y. Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Исполните программу в пошаговом режиме с трассировкой процедуры и проследите за изменениями значений переменных в основной программе и в процедуре, обратите внимание на передачу значений при вызове процедуры ее формальным параметрам, а также на то, что в процессе выполнения процедуры и по ее окончании значения переменных A и B не изменялись.

Параметры-переменные

Параметры-переменные используются для определения результатов выполнения процедуры и в списке формальных параметров перечисляются после зарезервированного слова var с обязательным указанием типа. Каждому формальному параметру, объявленному как параметр-переменная, должен соответствовать фактический параметр в виде переменной соответствующего типа, например:

```
procedure Example(var M, N : integer; var Y : real);
```

Если формальный параметр определен как параметр-переменная, то при вызове процедуры передается сама переменная, а не ее копия, и изменение параметра-переменной приводит к изменению фактического параметра в вызывающей программе. Следовательно, исходные данные в процедуру из программы могут передаваться как через параметры-значения, так и через параметры-переменные, а результаты работы процедуры возвращаются в вызывающую программу только через параметры-переменные.

Пример программы, использующей параметры-переменные:

```
program Sum_Sub_Square;
var
  A, B : real;
  SumAB, SubAB : real;
  {Процедура с параметрами-переменными Sum, Sub}
  procedure Sum_Sub(X, Y : real; var Sum, Sub : real);
  begin
    Sum := X * X + Y * Y;
    Sub := X * X - Y * Y;
  end;
begin
  {Конец процедуры}
  {Начало основной программы}
  A := 1.5;
  B := 3.4;
  Sum_Sub(A, B, SumAB, SubAB); {Вызов процедуры с передачей ей фактических
  параметров-значений A, B и параметров-переменных SumAB, SubAB}
  WriteLn('Сумма квадратов чисел 'A' и 'B' = 'SumAB);
  WriteLn('Разность квадратов чисел 'A' и 'B' = 'SubAB);
end.
```

Упражнение 4. Запустите интегрированную среду программирования. Введите текст программы Sum_Sub_Square и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После того как компиляция выполнится успешно, задайте для просмотра в окне отладчика величины A, B, Sum, Sub, SumAB, SubAB. Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Выполните программу в пошаговом режиме с трассировкой процедуры и проследите за изменениями значений переменных в основной программе и в процедуре, обратив внимание на передачу значений при вызове процедуры ее формальным параметрам,

а также на то, что значения переменных SumAB и SubAB в основной программе изменились в процессе выполнения процедуры.

Параметры-процедуры и параметры-функции

В качестве расширения стандартного Pascal, Turbo Pascal позволяет рассматривать процедуры и функции как объекты, которые можно присвоить переменным и которые могут выступать в качестве параметров; это становится возможным за счет использования процедурных типов.

Как только процедурный тип определен, можно объявлять переменные этого типа. Такие переменные называются *процедурными переменными*. Они могут использоваться в качестве формальных параметров при вызове процедур и функций. Подобно тому как целочисленной переменной можно присвоить целочисленное значение, процедурной переменной можно присвоить процедурное значение. Такое значение может, конечно, быть и другой процедурной переменной, но может также быть идентификатором процедуры или функции. В этой ситуации объявление процедуры или функции можно рассматривать как особый вид объявления константы, значением которой является процедура или функция.

Как и во всех других операциях присваивания, переменные в левой и правой части оператора присваивания должны быть совместимыми по присваиванию. Для того чтобы считаться совместимыми по присваиванию, процедурные типы должны иметь одинаковое число параметров, параметры в соответствующих позициях должны быть тождественных типов; наконец, типы результатов функций должны совпадать. Кроме совместимости типов, процедура или функция должны удовлетворять следующим требованиям, если они присваиваются процедурной переменной: они должны быть объявлены с директивой *far* (использование дальнего типа вызова подпрограмм) и откомпилированы в состоянии *{F+}*; они не должны являться стандартной процедурой или функцией, вложенной процедурой или функцией, *inline*-процедурой или функцией, *interrupt*-процедурой или функцией.

При использовании параметров-процедур или параметров-функций в списке перед соответствующими формальными параметрами указывается зарезервированное слово *procedure* или *function*, например:

```
procedure Exampl(K,L:integer; var M : real; procedure Prob; function Step: real);
```

В списке формальных параметров процедуры Exampl:

K, L — параметры-значения;

M — параметр-переменная;

Prob — параметр-процедура;

Step — параметр-функция, результатом выполнения которой является значение вещественного типа.

При вызове подпрограммы на место формальных параметров-процедур и параметров-функций осуществляется подстановка имен соответствующих фактических процедур или функций. При этом если процедуры и функции, фигурирующие в качестве фактических параметров, имеют, в свою очередь, параметры, они могут быть только параметрами-значениями.

Параметры процедурного типа особенно полезны в ситуациях, когда над несколькими процедурами или функциями выполняются общие действия. Примером использования параметров-функций может служить программа, которая с помощью одной и той же процедуры печати таблицы выводит на экран три таблицы арифметических функций (сложения, умножения и произведения суммы на разность чисел), каждая из которых выполняется отдельной функцией.

```
program Demo_Tab1;
{Описание процедурного типа Func — целой функции двух аргументов целого типа}
type
  Func = function(X,Y : integer) : integer;
{$F+} {Директива компилятору на использование дальнего типа вызова подпрограмм}
      {Описание функции сложения двух целых чисел}
      function Add(X,Y : integer) : integer;
      begin
        Add:=X+Y;
      end;
      {Описание функции умножения двух целых чисел}
      function Mult(X,Y : integer) : integer;
      begin
        Mult:=X*Y;
      end;
      {Описание функции умножения суммы на разность двух целых чисел}
      function Funny(X,Y : integer) : integer;
      begin
        Funny:=(X+Y)*(X-Y);
      end;
{$F-}
{Описание процедуры вывода на экран таблицы для двух чисел от 1 до 10. вид
арифметической операции задается значением параметра-функции Operation}
procedure Type_Tab1(W,H : integer; Operation : Func);
var
  X,Y : integer;
begin
  for I:=1 to H do
    begin
      for X:=1 to W do
        Write(Operation(X,Y):5);
        WriteLn;
      end;
    end;
end;
```



```

begin                                {Начало главной программы}
  Type_Tab1(10,10,Add): {Вызов процедуры для печати таблицы сложения}
  Type_Tab1(10,10,Mult): {Вызов процедуры для печати таблицы умножения}
  Type_Tab1(10,10,Funny): {Вызов процедуры для печати таблицы произведений суммы
чисел от 1 до 10 на их разность}
end.

```

ПРИМЕЧАНИЕ

{\$F+} — директива компилятора Turbo Pascal на использование дальнего (far) типа вызова для корректной обработки вызова процедуры Type_Tab1 с параметрами-функциями

В данной программе процедура Type_Tab1 представляет собой общее действие, выполняемое над параметрами-функциями Add, Mult, Funny. После запуска программы сначала вызывается процедура Type_Tab1 для фактических параметров 10, 10 и Add, в результате чего формальным параметрам X и Y присваиваются значения 10 и 10, а формальному параметру Operation процедурного типа Func присваивается имя фактической функции Add. В результате этого на экран выводится таблица сложения чисел от 1 до 10. Затем процедура Type_Tab1 вызывается для фактических параметров 10, 10 и параметра-функции Mult, в результате чего на экран выводится таблица умножения чисел от 1 до 10. Аналогично, вызов процедуры Type_Tab1 с параметрами 10, 10 и Funny дает в результате на экране таблицу произведения суммы на разность чисел от 1 до 10.

Упражнение 5. Запустите интегрированную среду программирования. Введите текст программы Demo_Tab1 и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции выполните программу в пошаговом режиме с трассировкой процедуры и проследите за вызовом функций вычисления суммы или произведения двух чисел или произведения их суммы и разности. Обратите внимание на выполнение оператора Write(Operation(X,Y):5): в зависимости от фактического значения параметра-функции Operation процедурного типа Func осуществляется вызов разных функций Add, Mult или Funny. Попробуйте удалить строку с директивой использования дальнего типа вызова или заключите в фигурные скобки описание процедурного типа Func и проследите за результатом. В случае появления ошибок нажатием клавиши F1 получите справку о причинах ошибки и рекомендации по ее исправлению.

Область действия параметров

При создании программ, использующих процедуры, следует учитывать, что все объекты (метки, константы, типы, переменные, процедуры и функции), которые описываются после заголовка процедуры, являются *локальными объектами* и доступны только в пределах этой процедуры, но недоступны в вызывающей программе. Все эти объекты создаются при входе в процедуру и уничтожаются при выходе из нее. Если одно и то же имя определено в

нескольких процедурах, вызываемых одной и той же программой, то в каждой процедуре этому имени соответствует свой локальный объект.

Все объекты, описанные в вызывающей программе, называются *глобальными* и являются доступными внутри процедур, вызываемых этой программой. Поэтому обмен данными между программой и вызываемой ею процедурой может производиться и через глобальные переменные. Если одно и то же имя определено и в программе, и в вызываемой ею процедуре, то ему соответствует глобальный объект, но внутри процедуры глобальный объект недоступен, он как бы экранируется (маскируется) локальным объектом с тем же именем.

В Turbo Pascal допускается любой уровень вложенности процедур и функций. Процедура, описанная в основной программе, в свою очередь, может иметь описания внутренних процедур и функций и т. д. При этом объекты, описанные в вызывающей процедуре, являются глобальными по отношению к вызываемой процедуре.

Можно схематически изобразить структуру блоков Pascal-программы, как показано на рис. 5.2.

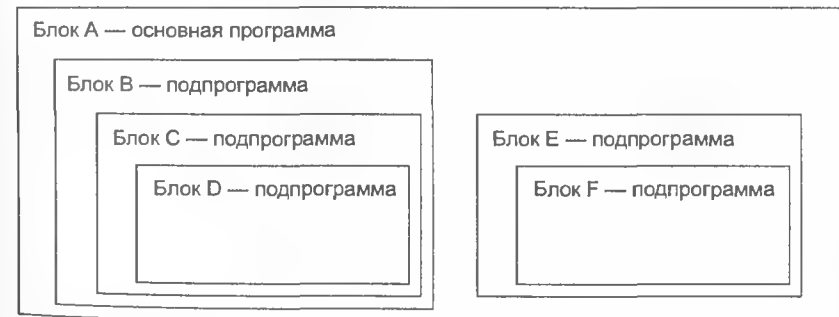


Рис. 5.2. Структура блоков Pascal-программы

Для доступа к объектам, описанным в разных блоках, требуется соблюдать следующие правила.

1. Имена объектов, описанных в блоке, считаются известными в пределах данного блока, включая и все вложенные блоки.
2. Имена объектов, описанных в блоке, должны быть уникальны в пределах данного блока и могут совпадать с именами объектов из других блоков.
3. Если в блоке описан объект, имя которого совпадает с именем объекта, описанного во внешнем блоке, то этот объект внешнего блока становится недоступным в данном блоке (он как бы экранируется одноименным объектом данного блока).

Если применить эти правила к предыдущей схеме, можно сказать, что объекты, описанные в блоке В, известны (видимы) кроме самого блока В еще и в бло-

ках C и D, но невидимы в блоке A. Объекты, описанные в блоке F, известны только в пределах этого блока. Например:

```
program Examp1_proc;
  procedure P;
    procedure A;
      var J : integer; {Локальная переменная J, является глобальной по отношению к процедуре B}
      procedure B;
        var J : integer; {Локальная переменная J, экранирует глобальную переменную J, описанную в вызывающей процедуре A}
      begin
        WriteLn(J);
      end;
    begin
      J:=1;
      B;          {Вызов процедуры B}
    end;
  begin
    A;            {Вызов процедуры A}
  end.
```

Иногда при вызове подпрограмм-функций возникают побочные эффекты, выражающиеся в том, что вносятся нежелательные изменения в значения глобальных переменных. Поэтому будьте внимательны при описании параметров-переменных, при выборе имен учитывайте наличие глобальных объектов с такими же именами.

В качестве примера с вложенными подпрограммами рассмотрим пример программы, определяющей количество сверхпростых чисел в натуральном ряду чисел, не превышающих 1000. Сверхпростым называется число, если оно простое, и число, полученное из исходного числа при записи цифр исходного числа в обратном порядке (перевертыш) тоже является простым. Например, 13 и 31 — сверхпростые числа.

```
{Подсчет количества сверхпростых чисел < 1000}
program Sverx_Prost;
  var X, K : integer; {Описание глобальных переменных X, K}
      {Функция проверки числа X на простое число}
  function Prost(Y : integer):boolean;
    var D, I : integer; {Описание локальных переменных D, I, Flag}
        Flag : boolean;
  begin
    D:=0; Flag:=False;
    for I:=2 to Y-1 do if Y mod I=0 then D:=D+1;
    if D=0 then Flag:=True;
    Prost:=Flag; {Нашли простое число, имеющее только два делителя: единицу и само это число}
  end;
```

```
{перевертыш простого числа}
function Povorot(Y:integer):integer;
  var S:integer; {Описание локальной переменной S}
begin
  if Y<10 then S:=Y;
  if (Y>10) and (Y<100) then S:=Y mod 10*10+Y div 10;
  if (Y>=100) and (Y<1000)
    then S:=Y mod 10*100+Y mod 100 div 10*10+Y div 100;
  Povorot:=S;
end;
begin {Начало главной программы}
  WriteLn(' 20.'Сверхпростые числа');
  K:=2; {2 и 3 — простые числа}
  for X:=4 to 999 do
    begin
      if Prost(X) then {Вызов функции определения простого числа X}
        if Prost(Povorot(X)) then {Вызов функции определения простого
числа для числа-перевертыша, полученного вычислением функции Povorot(X)
от простого числа}
          begin
            WriteLn(X); {Печать сверхпростого числа}
            K:=K+1;
          end;
    end;
  WriteLn('Всего найдено 'K.' сверхпростых чисел < 1000');
end.
```

Упражнение 6. Проанализируйте текст программы, обратив особое внимание на применение функций определения простого числа, функцию получения перевертыша и вложенный вызов функции Prost(Povorot(X)), при котором переменные D, I, Flag, локальные для функции Prost, являются глобальными для функции Povorot.

Запустите интегрированную среду программирования. Введите текст программы Sverx_Prost и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции задайте для просмотра в окне отладчика переменные X, K, I, D, S, Flag. Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Выполните программу в пошаговом режиме с трассировкой функции и проследите за изменениями значений переменных в основной программе и в функциях, обратите внимание на передачу значений при вызове функции от фактических параметров основной программы формальным параметрам функции, на возвращение вычисленных функцией значений в точку вызова главной программы, а также на характер изменения значений параметров локальных переменных I, D и глобальных переменных X, K.

Рекурсии

Рекурсия — это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе. Примером программы с использованием рекурсии может служить программа вычисления факториала числа. (Факториалом натурального числа n называют произведение чисел $1 \cdot 2 \cdot \dots \cdot n$.)

```
{Вычисление факториала числа n с использованием рекурсивной функции}
program Demo_Rekurs;
  var N : integer;
      F : longint;
  function Fakt(N : integer): longint; {Описание функции. N — формальный
  параметр-значение типа integer, результат выполнения функции типа longint}
  begin
    {Начало вычисления функции}
    if N=1 then Fakt:=1 {Проверка условия завершения рекурсии}
    else Fakt:=N*Fakt(N-1) {Рекурсивное вычисление N!}
  end;
begin
  {Начало главной программы}
  WriteLn('Введите число N >');
  Read(N);
  F:=Fakt(N); {Вызов функции для фактического параметра N}
  WriteLn('Для числа '.N.' значение факториала ='.F);
end.
```

После запуска программы на экран выводится запрос «Введите число N >», затем с клавиатуры считывается значение целого числа N и в выражении $F:=Fakt(N)$ вызывается функция Fakt с параметром-значением N. В подпрограмме-функции вычисления факториала проверяется условие $N = 1$. Если оно выполняется, то функции Fakt присваивается значение 1, на этом выполнение подпрограммы-функции завершается. Если условие $N = 1$ не соблюдается, то выполняется вычисление произведения $N * Fakt(N - 1)$. Вычисление произведения носит рекурсивный характер, так как при этом осуществляется вызов функции Fakt(N - 1), значение которой вычисляется, в свою очередь, посредством вызова функции Fakt, параметром которой также является функция Fakt, и т. д., до тех пор, пока значение формального параметра не станет $N = 1$. Так как базовая часть описания рекурсивной функции Fakt определяет значение Fakt для $N = 1$ равным единице, то рекурсивные вызовы функции Fakt больше не выполняются, а наоборот, выполняется вычисление функции Fakt для чисел, возрастающих от 1 до N, причем функция Fakt всякий раз возвращает значение, равное произведению очередного k-го числа на факториал от (k-1)-го числа. Последнее возвращение результата вычисления функции Fakt присвоит переменной F значение произведения всех чисел от 1 до N, т. е. факториал числа N.

Итак, при выполнении рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к

нижнему уровню последовательно до тех пор, пока, наконец, не будет получено тривиальное решение поставленной задачи. В нашем примере решение при $N = 1$ тривиально, т. е. $Fakt = 1$. Затем осуществляется возврат на верхний уровень с последовательным вычислением значения функции Fakt.

Упражнение 7. Запустите интегрированную среду программирования. Введите текст программы Demo_Rekurs и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции задайте для просмотра в окне отладчика переменные N, F. Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Выполните программу в пошаговом режиме с трассировкой функции и проследите за изменением значения переменной N при рекурсивных вызовах функции Fakt.

Следует учитывать, что использование рекурсивной формы организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызывать переполнение стека. Это объясняется тем, что при каждом входе в подпрограмму ее локальные переменные размещаются в особом образом организованной области памяти, называемой программным стеком.

Нетрадиционное использование подпрограмм

В Turbo Pascal есть случаи нетрадиционного объявления подпрограмм, когда в объявлении процедуры содержится директива interrupt (прерывание), external (внешняя) или inline (встроенная) или вместо блока в объявлении процедуры или функции написано forward (опережающая).

Interrupt (прерывание)

Объявление процедуры может содержать директиву interrupt перед блоком, и тогда процедура рассматривается как процедура прерывания. *Прерыванием* называется временное прекращение процесса, вызванное событием, внешним по отношению к этому процессу. Необходимость в процедурах обработки прерывания возникает, когда программист решает определить собственные алгоритмы реакции на прерывание операционной системы, отменив при этом стандартные процедуры. Отметим, что процедуры обработки прерывания нельзя вызывать с помощью операторов процедур и что каждая процедура обработки прерывания должна задавать список параметров в точности так, как это показано ниже:

```
procedure MyInt(Flags. CS. IP. AX. BX. CX. DX. SI. DI. DS. ES. BP:Word);
interrupt;
```

ПРИМЕЧАНИЕ

CS, IP, AX, BX, CX, DX, SI, DI, DS, ES, BP — регистры процессора.

Внешние объявления (external)

Внешние объявления позволяют связывать скомпилированные отдельно процедуры и функции, написанные на языке ассемблера. С помощью директивы `{L имя файла}` внешнюю программу можно связать с программой или модулем, написанным на Pascal.

Приведем следующие примеры объявлений внешних процедур:

```
procedure MoveWord(var Source, Dest; Count: Word); external;
procedure FillLong(var Dest; Data: Longint; Count: Word); external
```

В тексте программы при объявлении внешних подпрограмм нужно задать директиву `{L}`, аргументом которой является имя OBJ-файла, содержащего код подключаемой подпрограммы, например: `{L BLOCK.OBJ}`

Assembler

Assembler-объявление позволяет написать процедуру или функцию на встроенном Ассемблере (язык программирования низкого уровня, близкий к машинному).

Inline (встроенная)

Директива `inline` позволяет записывать инструкции в машинном коде, не используя блок операторов. При вызове обычной процедуры компилятор создает код, в котором параметры процедуры помещаются в стек, а затем для вызова процедуры генерируется инструкция `call`. Когда вызывается внутренняя процедура, компилятор генерирует код из директивы `inline` вместо `call`. Таким образом, `inline`-процедура «расширяется» при каждом обращении к ней аналогично макрокоманде на языке ассемблера. (Макрокоманда (`macro`) — предложение языка программирования, вместо которого компилятор при трансляции записывает несколько машинных команд.)

Опережающие объявления (forward)

Помимо прямых рекурсий, рассмотренных ранее, рекурсивный вызов может быть и косвенным, т. е. одна процедура вызывает другую процедуру, которая, в свою очередь, вызывает первую процедуру. А поскольку до вызова процедуры она обязательно должна быть описана, то используется опережающее объявление: процедура содержит описание только своего заголовка, вслед за которым ставится зарезервированное слово `forward` (опережающий), а описание текста процедуры помещается далее в тексте программы уже без повторения описания списка формальных параметров и называется *определяющим* объявлением.

Опережающее объявление и определяющее объявление должны находиться в одной и той же части объявления процедур и функций. Между ними могут быть объявлены другие процедуры и функции, и они могут вызывать процедуру с опережающим объявлением. Таким образом, возможно использование взаимной рекурсии.

Определяющее объявление процедуры может иметь тип `external` или `assembler`. Однако оно не может быть `near`-, `far`-, `inline`- или другим `forward`-объявлением. Определяющее объявление также не может содержать директивы `interrupt`, `near` или `far`. Опережающие объявления не допускаются в интерфейсной части модуля. Например:

```
program Demo_Kos_Rekurs;
  var a,b : real;
  procedure P1(x : real); forward; {Опережающее объявление процедуры}
  procedure P2(y : real);
  begin
    P1(a); {Вызов процедуры P1}

  end;
  procedure P1: {Текст процедуры без описания предварительно описанных параметров}
  begin
    P2(a); {Вызов процедуры P2}

  end;
begin
  P2(a); P1(b); {Вызов процедур P2 и P1}

end.
```

Как видно из этого примера, опережающее объявление процедуры `P1` позволило использовать обращение к ней из процедуры `P2`, так как при трансляции компилятору до вызова процедуры `P1` из опережающего объявления становятся известными ее формальные параметры и он может правильно организовать ее вызов. В самом теле процедуры `P1` уже нет необходимости описывать параметры, так как это было сделано при опережающем описании.

Контрольные вопросы и задания

Вопросы

1. Что называется подпрограммой? В чем состоит сходство и различие подпрограмм-процедур и подпрограмм-функций в языке Turbo Pascal?
2. В чем различие между стандартными и определенными пользователем подпрограммами? Приведите примеры.
3. Запишите синтаксическую диаграмму определения процедуры, функции.
4. Опишите последовательность событий при вызове процедуры или функции.
5. В каких случаях в программе указывается директива компилятору `{I}`?

6. Что называется параметром, и каково его назначение? Формальные, фактические параметры, их взаимосвязь.
7. Для чего используется исполнение программы в пошаговом режиме с заходом в процедуры, и как это осуществить?
8. Каковы отличия параметров-значений от параметров-переменных, особенности их описания и применения.
9. Каковы особенности параметров-процедур и параметров-функций?
10. Чем отличаются локальные и глобальные параметры? Какова область их действия?
11. Что такое рекурсия?
12. В каких случаях требуется предварительное или внешнее описание подпрограмм? Каковы особенности использования подпрограмм с предварительным описанием?
13. Даны фрагменты программ:

1) var

```
c,d : integer;
procedure P(x,y : integer);
begin
  y:= x+1;
end;
```

2) var

```
c,d : integer;
procedure Q(x : integer; var y : integer);
begin
  y:= x+1;
end;
```

3) var

```
c,d : integer;
procedure R(var x,y : integer);
begin
  y:= x+1;
end;
```

- а) для каждой из этих процедур указать, какие из ее параметров являются параметрами-значениями, а какие — параметрами-переменными.
- б) определить, что будет выдано на печать в следующих случаях:

```
c:=2; d:=0; P(sqr(c)+c,d); WriteLn(d);
c:=2; d:=0; Q(sqr(c)+c,d); WriteLn(d);
```

Почему при изменении в процедуре параметра-значения соответствующий фактический параметр не меняет своего значения? Что надо сделать, чтобы он менял значение?

- в) допустимы ли обращения $R(\text{sqr}(c)+c,d)$ и $R(c,d)$? Почему невыгодно объявлять параметр, не меняющийся в процедуре, параметром-переменной?

Задания

1. Напишите программу вычисления среднего геометрического модулей двух введенных с клавиатуры целых чисел X и Y . Программа должна использовать цикл WHILE DO. Условие выхода из цикла — значение числа, равное 999. (Средним геометрическим называется корень квадратный из произведения двух чисел.)
2. Напишите программу вычисления выражения $y = (\text{tg}(X) + \sin(X)) * e^{\cos(X)}$ при $x = 3.7$. Результат выведите в формате 5:2.
3. Напишите программу, которая находит первое отрицательное число последовательности: $A = \text{Sin}(i/100)$. $i = 1,2,3...$
4. Напишите программу, которая с помощью функции Chr выводит на экран кодовую таблицу ASCII. После задержки в 5 секунд очистите экран.
5. Напишите программу, выводящую на экран 10 строк по 5 случайных чисел из диапазона 0..36.
6. С помощью цикла for и функции Odd напишите программу, выводящую все нечетные числа из диапазона 1..100.
7. Напишите программу, которая по значениям двух катетов вычисляет гипотенузу и площадь треугольника.
8. Напишите программу вычисления расстояния между двумя точками с заданными координатами $X1, Y1, X2, Y2$.
9. Напишите процедуру-заставку к программе вычисления математических функций в виде

```
*****
*                                     *
*                               Программа                               *
*      вычисления математических функций      *
*                               Автор: Смирнов А.П.                               *
*****
```

Заставка должна выводиться на очищенный экран, оставаться на экране в течение 3 секунд с последующей очисткой экрана. Вызовите процедуру Zastavka в начале программы.

10. Напишите функцию возведения в степень по формуле: $A^B = \text{Exp}(\text{Ln}(A) * B)$ и используйте ее в программе для возведения в 4-ю степень вещественного числа 2,87.
11. Оформите процедуру проверки права пользователя работать с программой. Используйте для этого пароль SCHOOL. В случае ввода неправильного пароля выход из программы осуществляется при помощи инструкции Halt.
12. Напишите программу вычисления функции Y по формуле:

$$y = (\ln^2(1+x) + \cos^e(x+1)),$$

$$\text{где } x = \begin{cases} kz^3, & \text{при } k < 1 \\ z(z+1), & \text{при } k \geq 1 \end{cases}$$

13. Напишите программу, состоящую из трех процедур и основной программы. Первая процедура организует ввод двух целых чисел X и Y , вторая вычисляет их сумму, третья выводит результат. Используйте эти процедуры в основной программе. Используйте X , Y как глобальные переменные. Эта программа послужит прообразом всех ваших будущих программ, так как в ней реализуется принцип работы любой системы: логически выделенные ввод, обработка и вывод результата.
14. Напишите программу вычисления для планет солнечной системы площади поверхности и длины экватора на основе известного радиуса. Формулы для планет будем считать шарообразной. Вычисление площади и длины экватора оформите в виде отдельных функций.
15. Напишите программу поиска максимального из четырех чисел с использованием подпрограммы поиска большего из двух.
16. Даны координаты вершин многоугольника $(x_1, y_1), (x_2, y_2), \dots, (x_{10}, y_{10})$. Определите его периметр (вычисление расстояния между вершинами оформить подпрограммой).
17. Вычислите сумму: $1! + 2! + 3! + \dots + n!$, используя функцию вычисления факториала числа $k!$.
18. Вычислите сумму простых, сверхпростых, совершенных чисел, не превосходящих заданного числа N .
19. Создайте программу вычисления числа сочетаний из N по M . Число сочетаний определяется по формуле $N!/(M!(N-M)!)$, где N — количество элементов перебора. Используйте подпрограмму вычисления факториала.
20. Определите наименьший общий делитель трех натуральных чисел.
21. Даны действительные числа s, t . Напишите программу вычисления выражения $f(t, -2s, 1.17) + f(2.2, t, s-t)$, где $f(a, b, c) = (2a - b - \sin(c)) / (5 + |c|)$.
22. Дано натуральное число N . Напишите программу, определяющую, есть ли среди чисел $n, n+1, \dots, 2n$ «близнецы», т. е. простые числа, разность между которыми равна 2. (Используйте процедуру распознавания простых чисел).
23. Создайте программу перевода двоичной записи натурального числа в десятичную.
24. Создайте программу сокращения дроби M/N , где M, N — натуральные числа.
25. Создайте программу для вычисления суммы квадратов простых чисел, лежащих в интервале (M, N) .

26. Создайте программу для подсчета числа четных цифр, используемых в записи N -значного числа M .
27. Создайте программу вычисления суммы трехзначных чисел, в десятичной записи которых нет четных цифр.
28. Создайте программу вывода на экран всех натуральных чисел, не превосходящих N и делящихся на каждую из своих цифр.
29. Создайте программу нахождения наименьшего натурального N -значного числа X ($X \geq 10$), равного утроенному произведению своих цифр.
30. Создайте программу подсчета числа всех натуральных чисел, меньших M , квадрат суммы цифр которых равен X .

Глава 6

Структурированные типы данных. Строки

Все простые типы данных, рассматривавшиеся в предыдущих главах, имеют два характерных свойства: неделимость (атомарность) и упорядоченность значений.

Составные, или *структурные*, типы данных, в отличие от простых, задают множества сложных значений с одним общим именем. Можно сказать, что структурные типы определяют некоторый способ образования новых типов данных на основе уже имеющихся. Таким образом, Turbo Pascal допускает образование структур данных произвольной сложности, позволяя тем самым достичь адекватного представления в программе тех данных, с которыми она оперирует.

Существует несколько методов структурирования, каждый из которых отличается способом обращения к отдельным компонентам и, следовательно, способом обозначения компонентов, входящих в структурные данные. По способу организации и типу компонентов в сложных типах данных выделяют следующие разновидности:

- регулярный тип (массивы);
- комбинированный тип (записи);
- файловый тип (файлы);
- множественный тип (множества);
- строковый тип (строки);
- объектный тип (объекты; введен в Turbo Pascal, начиная с версии 6.0).

ПРИМЕЧАНИЕ

В Delphi 6 появилась возможность на основе *variant* создавать собственные типы данных.

В отличие от простых типов данных, данные структурированного типа характеризуются множественностью образующих этот тип элементов, т. е. переменная или константа структурированного типа всегда имеет несколько

компонентов. Каждый компонент, в свою очередь, может принадлежать структурированному типу, т. е. возможна вложенность типов.

Описание строкового типа

Изучение данных структурированного типа начнем со строкового типа данных (строка). *Строка* — это последовательность символов ASCII. При использовании в выражениях строка заключается в апострофы. Количество символов в строке (длина строки) может динамически изменяться в пределах от 0 до 255. Для определения данных строкового типа используется идентификатор *string*, за которым следует заключенное в квадратные скобки значение максимально допустимой длины строки данного типа. Если это значение не указывается, то по умолчанию длина строки принимается равной 255 байтам.

Переменную строкового типа можно определить через описание типа в разделе определения типов или непосредственно в разделе описания переменных. Строковые данные могут использоваться в программе также в качестве констант. Не допускается применение строковых переменных в качестве селектора в операторе *case*.

Определение строкового типа устанавливает максимальное количество символов, которое может содержать строка.

Формат:

```
type
  <имя типа> = string [максимальная длина строки];
var
  <идентификатор...> : <имя типа>;
```

Переменную типа *string* можно задать и без описания типа:

```
var
  <идентификатор...> : string [максимальная длина строки];
```

Пример:

```
const
  Address = 'ул. Переверткина. 25': {Строковая константа}
type
  Flot = string[125];
var
  Fstr : Flot;           {Описание с заданием типа}
  St1 : string;          {По умолчанию длина строки = 255}
  St2, St3 : string[50];
  Nazv : string[255];    {Ошибка, длина Nazv превышает 255}
```

Строка в языке Turbo Pascal трактуется как цепочка символов. Для строки из *N* символов отводится *N + 1* байт: *N* байт для хранения символов строки, а один байт — для значения текущей длины строки.

К любому символу в строке можно обратиться, указав его номер. В самом начале строки (под нулевым номером) расположен байт, содержащий значение текущей длины строки.



Поэтому для определения объема памяти в байтах, требуемой для размещения строки, к значению ее максимальной длины прибавляется 1. Например, для размещения в памяти переменных *Fstr*, *St1*, *St2* требуется соответственно 126, 35 и 51 байт. Рассмотрим структуру размещения строки в памяти на следующем примере. Пусть *M* — максимальная длина строки, *L* — текущая длина, *A* — ячейка памяти. Тогда:

A — содержит величину текущей длины;

A + 1 — первый символ строки;

A + *L* — последний значащий символ;

A + *L* + 1

... — незанятые ячейки памяти

A + *M*

Строковые выражения

Выражения, в которых операндами служат строковые данные, называются *строковыми*. Они состоят из строковых констант, переменных, указателей функций и знаков операций. Над строковыми данными допустимы операция сцепления и операции отношения.

Операция сцепления (+) применяется для объединения нескольких строк в одну результирующую строку. Например:

Выражение	Результат
'A'+'T'+' '+'386'	'AT 386'
'Turbo'+'Pascal'+'7.0'	'Turbo Pascal 7.0'

Следует учитывать, что в операциях сцепления длина результирующей строки не должна превышать 255.

Операции отношения (=, <, >, <=, >=) производят сравнение двух строковых операндов и имеют более низкий приоритет, чем операция сцепления, т. е. сначала всегда выполняются все операции сцепления, если они есть, и лишь потом реализуются операции отношения. Сравнение строк производится слева направо до первого несовпадающего символа, и большей считается та

строка, в которой первый несовпадающий символ имеет больший номер в таблице ASCII. Результат выполнения операций отношения над строковыми операндами всегда имеет булевский тип и принимает значение True, если выражение истинно, и False, если выражение ложно. Например:

Выражение	Результат
'MS-DOS' < 'MS-Dos'	True
'program' > 'PROGRAM'	True

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки считаются равными, если они полностью совпадают по длине и содержат одни и те же символы. Например:

Выражение	Результат
'Принтер' > 'Принтер'	True
'Intell' = 'Intell'	True

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания (:=).

Пример:

```

Группа учащихся';
Str1 + ' школы-лицея'
      чаров А.А.'

```

Если значение переменной после выполнения операции присваивания превышает по длине максимально допустимую при описании величину, все лишние символы справа отбрасываются, например:

Описание A	Выражение	Значение A
A: string[6]	A := 'ГРУППА 1';	'ГРУППА'
A: string[8]	A := 'ГРУППА 1';	'ГРУППА 1'
A: string[2]	A := 'ГРУППА 1';	'ГП'

Допускается смещение в одном выражении операндов строкового и литерного типа. Если при этом литерной переменной присваивается значение строкового типа, длина строки должна быть равна единице, иначе возникает ошибка.

К отдельным символам строки можно обратиться по номеру (индексу) данного символа в строке. Индекс определяется выражением целочисленного типа, которое записывается в квадратных скобках сразу за идентификатор строковой переменной или константы. Например, выражения *Str2*[1+2] и *Str2*[7] обеспечат доступ к третьему ('Д') и седьмому ('З') символам последнего значения переменной *Str2* в приведенном выше фрагменте.

При помощи записи `Str2[0]` можно получить доступ к нулевому байту, держащему значение текущей длины строки. Значение нулевого байта должно превышать 255, но нарушение этого правила не вызывает программного прерывания, так как директива компилятора R по умолчанию находится в пассивном состоянии `{R-}`. Для обеспечения строгого контроля диапазоном допустимых значений индекса следует перевести директиву в активное состояние `{R+}`. В этом случае компилятор активизирует дополнительные команды для проверки правильности диапазона. Обычно активный режим R устанавливается на стадии отладки программ.

Для обработки строковых данных можно использовать специальные процедуры и функции.

Строковые процедуры и функции

`Delete (St,Poz,N)` — удаление N символов строки St, начиная с позиции Poz. Если значение Poz > 255, возникает программное прерывание.

Значение St	Выражение	Результат
'абвгде'	<code>Delete(Str, 4, 2);</code>	'абве'
'река Волга'	<code>Delete(Str, 1, 5);</code>	'Волга'

`Insert (Str1, Str2, Poz)` — вставка строки Str1 в строку Str2, начиная с позиции Poz. Например:

```
var
  S1, S2: string[11];

S1 := 'EC';
S2 := 'IBM 1841';
Insert(S1, S2, 4);
```

В результате выполнения последнего выражения значение строки S2 станет равным 'IBM EC 1841'.

`Str (IBR,St)` — преобразование числового значения величины IBR и помещение результата в строку St. После IBR может записываться формат, аналогичный формату вывода. Если в формате указано недостаточное для вывода количество разрядов, поле вывода автоматически расширяется до нужной длины.

Значение IBR	Выражение	Результат
1500	<code>Str(1500, St)</code>	'_1500'
4.8E+03	<code>Str(10, St)</code>	'____4800'
76854	<code>Str(-1, St)</code>	'-76854'

`Val (St,IBR,Code)` — преобразует значение St в величину целочисленного и вещественного типа и помещает результат в IBR. Значение St не должно со-

держивать незначащих пробелов в начале и в конце. Code — целочисленная переменная. Если операция преобразования не привела к ошибке, то значение Code равно нулю. В случае ошибки (например, литерное значение переводится в цифровое) Code будет содержать номер позиции первого ошибочного символа, а значение IBR будет не определено.

Значение St	Выражение	Результат
'1450'	<code>Val(St,IBR,Cod)</code>	Code=0
'14.2E+02'	<code>Val(St,IBR,Cod)</code>	Code=0
'14.2A+02'	<code>Val(St,IBR,Cod)</code>	Code=5

`Copy (St,Poz,N)` — выделяет из St подстроку длиной N символов, начиная с позиции Poz. Если Poz > Length(St), то результатом будет пробел; если Poz > 255, то возникнет ошибка. Функция Length описана ниже. Poz, N — целочисленные выражения.

Значение St	Выражение	Результат
'ABCDEFG'	<code>Copy(St, 2, 3)</code>	'BCD'
'ABCDEFG'	<code>Copy(St, 4, 10)</code>	'DEFG'

`Concat (Str1,Str2,...,StrN)` — выполняет сцепление строк Str1, Str2, StrN в том порядке, в каком они указаны в списке параметров. Сумма символов всех сцепленных строк не должна превышать 255, например:

Выражение	Результат
<code>Concat('AA','XX','Y')</code>	'AAXXY'
<code>Concat('Индекс','394063')</code>	'Индекс 394063'

`Length (St)` — вычисляет текущую длину в символах строки St. Результат имеет целочисленный тип, например:

Значение St	Выражение	Результат
'123456789'	<code>Length(St)</code>	9
'System 370'	<code>Length(St)</code>	10

`Pos (Str1,Str2)` — обнаруживает первое появление в строке Str2 подстроки Str1. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки Str1. Если в Str2 не содержится подстрока Str1, то результат равен 0.

Значение Str1	Выражение	Результат
'abcdef'	<code>Pos('de', Str1)</code>	4
'abcdef'	<code>Pos('r', Str1)</code>	0

UpCase (Ch) — преобразует строчную букву в прописную. Параметр и результат имеют литерный тип. Обрабатываются буквы только латинского алфавита, например:

Значение	Выражение	Результат
'd'	UpCase(Ch)	'D'
'w'	UpCase(Ch)	'W'

Упражнения

Упражнение 1. Составим программу, которая после ввода строки строчных латинских букв заменяет их на прописные. Решение данной задачи можно разделить на три самостоятельные части: ввод строки строчных латинских букв, преобразование букв в прописные, вывод полученной строки из прописных букв. Назовем вводимую строку символов *S* и опишем ее как переменную строкового типа:

```
var
  S: string;
```

Для обращения к символу, входящему в строку, введем переменную *I* типа *byte*, так как в строке может быть не более 255 символов. Для ввода строки используем стандартную функцию *Readln(S)*, для вывода полученной строки из прописных букв используем стандартную функцию *Writeln(S)*. Для преобразования символов строки из строчных в прописные введем подпрограмму-процедуру *UpChar*.

```
program DemoUpper; {Преобразование строчных букв в ПРОПИСНЫЕ}
var
  S: string; {Описание S — строки переменной длины}
begin
  Write('Введите исходную строку: ');
  Readln(S); {Ввод исходной строки}
  UpChar(S); {Преобразование символов строки из строчных в прописные}
  Writeln(S); {Вывод выходной строки}
end.
```

При разработке подпрограммы-процедуры *UpChar(S)*, выполняющей преобразование символов строки из строчных в прописные, используем стандартную функцию *UpCase(S[I])* для преобразования строчной латинской буквы в прописную. Действие подпрограммы заключается в том, что, начиная с 1-го символа и до конца строки строчная буква заменяется прописной. Так как эта операция повторяется, то будет удобно записать ее в виде цикла *for*, параметр которого будет изменяться от 1 до величины длины строки, которую вычислит стандартная функция *Length(S)*. Текст процедуры будет таким:

```
procedure UpChar(var S: string);
var I: byte; {Локальная переменная I — номер очередного символа в строке}
```

Упражнения

```
for I = 1 to Length(S) do {Просматривая с 1-й до последней буквы строки}
  S[I] := UpCase(S[I]); {Преобразовать очередной символ}
```

В результате получится следующая программа:

```
program DemoUpper; {Преобразование строчных букв в ПРОПИСНЫЕ}
var
  S: string; {Описание S — строки переменной длины}
  I: byte; {Локальная переменная I — номер очередного символа в строке}
begin
  Write('Введите исходную строку: ');
  Readln(S); {Ввод исходной строки}
  UpChar(S); {Вызов процедуры преобразования символов строки из строчных
  в прописные с передачей ей параметра-переменной S}
  Writeln(S); {Вывод выходной строки}
end.
```

Запустите интегрированную среду программирования. Введите текст программы *DemoUpper* и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы, задавая строки текста в нижнем латинском регистре.

Упражнение 2. Изменим программу так, чтобы она во введенном слове подсчитывала число букв «а» и заменяла их буквами «б». Так как операции ввода-вывода строки аналогичны, то можно сделать вывод о том, что достаточно заменить процедуру *UpChar* на процедуру *ChangeChar* и поменять соответствующий вызов.

В процедуре *ChangeChar* будем просматривать строку с целью поиска буквы «а», что можно организовать с помощью цикла *while* и стандартной функции *Pos('a', S)*. Как только функция *Pos* обнаруживает первое появление в строке *S* подстроки «а», она возвращает результат — номер позиции буквы «а». Счетчик найденных букв «а» увеличивается на единицу, а в эту позицию вписывается буква «б» и так продолжается до тех пор, пока в строке есть буквы «а». Текст процедуры может выглядеть следующим образом:

```
procedure ChangeChar(var S: string);
var N: byte;
begin
  N := 0; {Обнуление числа букв "а"}
  while Pos('a', S) > 0 do {Если найдена буква "а". то}
  begin
    S[N+1] := 'b'; {Увеличить счетчик букв "а" на 1}
    S := S[N+1..Length(S)];
  end;
```

```

S[Pos('a', S)] := 'b'; {Записать в позицию буквы "a" букву "b"}
end;
Writeln ('В слове было ', N, ' букв "a"');
end;

```

С использованием процедуры ChangeChar текст программы, подсчитывающей число букв «а» во введенной строке и заменяющей их буквами «б», будет таким:

```

program Change_Letter; {Подсчет и замена букв "a" на "b"}
var
  S: string;
  N: integer;
  procedure ChangeChar(var S: string); {Процедура замены буквы "a" на "b"}
  var N: integer;
  begin
    N := 0; {Обнуление числа букв "a"}
    while Pos('a', S) > 0 do {Если найдена буква "a", то}
      begin
        N := N + 1; {Увеличить счетчик букв "a" на 1}
        S[Pos('a', S)] := 'b'; {Записать в позицию буквы "a" букву "b"}
      end;
    Writeln ('В слове было ', N, ' букв "a"');
  end;
begin {Основная программа}
  Write('Введите исходную строку ');
  Readln(S);
  ChangeChar(S); {Вызов процедуры замены "a" на "b"}
  Writeln('Получилась строка ', S);
end;

```

Запустите интегрированную среду программирования. Введите текст программы Change_Letter и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы с отладкой в пошаговом режиме и проследите за значением переменной S.

Упражнение 3. Составим программу, которая запрашивает две строки по четыре символа, состоящие из цифр. В случае ввода символов, отличных от цифр, должно отображаться соответствующее сообщение, а программа должна завершаться. Программа объединяет введенные строки, затем преобразует исходные строки в числа, подсчитывает их сумму, преобразует результат в строку и печатает строки, полученные в результате объединения и преобразования суммы чисел в строку. Работу программы можно представить в виде трех самостоятельных фрагментов: ввод первой и второй строк и преобразование их в число, вывод на экран результата объединения строк и суммирование чисел. Вывод на экран результатов организуем с помощью стандартных процедур вывода Writeln, а для ввода строк и преобразования их в числа создадим процедуру Inp_Str. Для передачи данных между процедурой и основной программой введем формальные параметры-переменные S типа string

и X типа integer. Для преобразования введенной строки в число применим стандартную функцию Val, а для анализа операции преобразования строки в число введем локальную переменную Code целого типа. После преобразования строки в число проверим значение переменной Code, если оно не равно 0, то значит в строке не все символы являются цифрами. Значение Code укажет позицию первого символа в строке, не являющегося цифрой. В этом случае напечатаем на экране сообщение об ошибке и укажем позицию неверно введенного символа в строке, после чего прервем работу программы, используя стандартную процедуру Halt. Текст процедуры Inp_Str будет выглядеть следующим образом:

```

procedure Inp_Str(var S: string; var X: integer); {Процедура ввода строки цифр
и преобразования строки в число}
var
  Cod: integer; {Результат преобразования строки в число}
begin
  Write('Введите строку цифр');
  Readln(S);
  Val(S, X, Cod); {Преобразование строки S в целое число X}
  if Cod <> 0 then {Если не все символы в строке являются цифрами}
    begin
      Writeln('Ошибка! В позиции ', Cod, ' введенной строки не цифра');
      Halt(1); {Прерывание программы}
    end;
end;

```

С учетом вышесказанного полный текст программы решения задачи будет записан следующим образом:

```

program Demo_Val_Str;
var S1, S2: string;
    X1, X2: integer;
  procedure Inp_Str(var S: string; var X: integer); {Процедура ввода строки цифр
и преобразования строки в число}
  var
    Cod: integer; {Результат преобразования строки в число}
  begin
    Write('Введите строку цифр');
    Readln(S);
    Val(S, X, Cod); {Преобразование строки S в целое число X}
    if Cod <> 0 then {Если не все символы в строке являются цифрами}
      begin
        Writeln('Ошибка! В позиции ', Cod, ' введенной строки не цифра');
        Halt(1); {Прерывание программы}
      end;
  end;
begin {Начало основной программы}
  Inp_Str(S1, X1); {Вызов процедуры ввода строки с фактическими
параметрами-переменными S1, X1}

```

```
Inp_Str(S2,X2); {Вызов процедуры ввода строки с фактическими
параметрами-переменными S2,X2}
Writeln('Результат склеивания строк -> ',Concat(S1,S2));
Writeln('Сумма введенных чисел= ',X1+X2);
end.
```

Запустите интегрированную среду программирования. Введите текст программы Demo_Val_Str и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы с отладкой в пошаговом режиме, отслеживая значения переменных S1, S2, X1, X2.

Упражнение 4. Создадим программу, определяющую, является ли введенное слово перевертышем. Перевертышем называется слово, которое одинаково читается как слева направо, так и справа налево, например: шалаш, казак.

Как видно из определения, для выяснения, является ли слово перевертышем, необходимо сравнивать 1-й и последний символ в строке, 2-й и предпоследний, 3-й и предпредпоследний символ, и т. д. до середины слова. Если в процессе сравнения будет установлено отличие сравниваемых символов, т. е. выясняется, что слово читается слева направо иначе, чем справа налево, значит можно сделать вывод, что это слово не является перевертышем. Если в процессе сравнения не будет выявлено отличия сравниваемых символов, значит это слово — перевертыш. Введем следующие переменные: для хранения слов — Word типа string с максимальным размером слов 30 символов и переменную I целого типа, указывающую номер позиции сравниваемого символа от начала строки. Заголовок программы можно будет записать следующим образом:

```
program Perev_Word;
var I : byte;
    Word : string[30];
```

Прежде всего программа должна выводить сообщение о вводе строки и считывать значение строки в переменную Word. Это можно записать следующим образом:

```
Write('Введите слово ');
Readln(Word);
```

Повторяющуюся операцию сравнения первого и последнего символа в строке, затем второго и предпоследнего и т. д., запишем с помощью цикла for, параметр которого, изменяясь от 1 до середины строки, будет указывать номер позиции символа от начала строки. Конечное значение параметра цикла установим равным середине слова, целочисленное значение которого вычислим, используя стандартные функции Trunc и Length. Заголовок оператора цикла запишем следующим образом:

```
for I:=1 to Trunc(Length(Word))/2 do
```

В теле цикла запишем оператор сравнения соответствующих символов:

```
if Word[I] = Word[Length(Word)-I+1] then
```

Если результат сравнения соответствующих очередных символов примет значение True, то данная строка не является перевертышем, и дальнейшее сравнение не имеет смысла. Выведем на экран сообщение, что данное слово не является перевертышем, и завершим цикл процедурой exit, которая передает управление в конец программы.

Если результат сравнения соответствующих очередных символов имеет значение False, то это свидетельствует о том, что данные символы одинаковы, но следует продолжить сравнение оставшихся. Параметр цикла увеличивается на 1, проверяется условие $I \leq \text{Trunc}(\text{Length}(\text{Word})/2)$ и если оно соблюдается, то тело цикла выполняется еще раз.

Если условие $I \leq \text{Trunc}(\text{Length}(\text{Word})/2)$ не выполняется, то цикл завершается, и управление передается на оператор Writeln('Перевертыш'). Программа завершает работу.

Полный текст программы можно представить таким образом:

```
program Perev_Word; {Является ли введенное слово перевертышем?}
var I : byte;
    Word : string[30];
begin
    Write('Введите слово ');
    Readln(Word);
    for I:=1 to Trunc(Length(Word))/2 do {Проверяем символы поочередно от начала
до середины слова}
    begin
        if Word[I] <> Word[Length(Word)-I+1] then {Если соответствующие символы не
одинаковы}
        begin
            Writeln('Неперевертыш');
            exit; {Выход из цикла и завершение программы, дальше не имеет смысла
сравнивать}
        end
    end;
    Writeln('Перевертыш');
```

Запустите интегрированную среду программирования. Введите текст программы Perev_Word и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы с отладкой в пошаговом режиме, отслеживая значения переменных Word, I, Word[I], Word[Length(Word)-I+1].

Упражнение 5. Создадим программу, которая обращает введенное слово, т. е. представляет символы в слове в обратном порядке, например: Петя — ятеП, мама — амам, программа — амаргморп. Алгоритм обращения слова можно представить в виде следующей циклической процедуры обмена соответствующих

символов: значение первого символа слова запоминается в некоторой переменной символьного типа Ch, затем на место первого символа записывается значение последнего символа, а на его место записывается значение первого символа, хранимое в переменной Ch, после этого аналогично выполняется обмен второго и предпоследнего символа, третьего и предпредпоследнего и т. д., пока мы не дойдем до середины слова. Далее обмен продолжать не нужно, иначе символы опять займут прежние места.

Исходя из этих рассуждений, введем следующие переменные: для хранения слова Word типа string с максимальным размером слов 30 символов и переменную I целого типа, указывающую номер позиции сравниваемого символа от начала строки, а также символьную переменную Ch для временного хранения значения символа при обмене. Заголовок программы можно будет записать следующим образом:

```
program Obr_Word;
var I : byte;
    Ch : char;
    Word : string[30];
```

Аналогично предыдущей программе, сначала выведем на экран запрос о вводе слова и считаем с клавиатуры значение слова в переменную Word. Этот фрагмент программы можно записать следующим образом:

```
Write('Введите слово ');
ReadLn(Word);
```

После ввода слова следует выполнение циклической процедуры обмена значений соответствующих символов (первого и последнего, второго и предпоследнего символов, третьего и предпредпоследнего и т. д. до середины слова), которое мы запишем в виде цикла for, параметр которого, изменяясь от 1 до середины слова, значение которого определяется результатом выражения $\text{Trunc}(\text{Length}(\text{Word})/2)$, указывает на позицию очередного символа в слове.

Перестановку соответствующих символов в слове с использованием символьной переменной Ch осуществим следующим образом:

```
Ch:=Word[I]; {переменной Ch присвоили значение I-го символа от начала слова}
Word[I]:=Word[Length(Word)-I+1]; {в I-ю позицию записали значение I-го символа
от конца (Length(Word)-I+1-го от начала) слова}
Word[Length(Word)-I+1]:=Ch; {в Length(Word)-I+1-ю позицию от начала слова
записали значение I-го символа, временно хранимое в переменной Ch}
```

В заключительной части программы выведем значение «обращенного» слова на экран с помощью стандартной процедуры вывода:

```
Write('Получилось слово ', Word);
```

Полный текст этой программы будет таким:

```
program Obr_Word;
var I : byte;
```

```
    Ch : char;
    Word : string[30];

begin
    Write('Введите слово ');
    ReadLn(Word);
    for I := 1 to Trunc(Length(Word)/2) do {Перебирая символы по-прежнему}
    begin {Обмениваем соответствующие символы}
        Ch:=Word[I];
        Word[I]:=Word[Length(Word)-I+1];
        Word[Length(Word)-I+1]:=Ch;
    end;
    Write('Получилось слово ', Word);
end.
```

Запустите интегрированную среду программирования. Введите текст программы Obr_Word и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы с отладкой в пошаговом режиме, отслеживая значения переменных Word, I, Word[I], Ch, Word[Length(Word)-I+1].

Задачу реверсирования (обращения текста) можно решить и с помощью программы с рекурсивным вызовом подпрограммы-функции, например:

```
program Reverse_Str;
type stroka = string[30];
var
    S, Rev_S : stroka;
function Reverse(Str:stroka) : stroka;
var
    FirstChar : char; {Первый символ в строке}
    OstatokStr : stroka; {Остаток строки после удаления первого символа}
begin
    if Length(Str)=1
    then := Str {Завершение рекурсии}
    else
    begin
        tChar:=Str[1];
        tStr:=Str[2..Length(Str)];
        tRev_S:=Reverse(tStr); {Рекурсивные вызовы функции}
        tRev_S:=Concat(tRev_S, tChar);
    end;
end;

begin {Начало основной программы}
    Write('Введите любой текст');
    ReadLn(S);
    Rev_S:=Reverse(S); {Вызов функции с параметром-значением S}
    WriteLn(Rev_S, ' есть перевернутое ', S);
end.
```

Упражнение 6. Изучите текст программы `Rever_Str`. Запустите интегрированную среду программирования. Введите текст программы `Rever_Str` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы с отладкой в пошаговом режиме, отслеживая значения переменных `St`, `FirstChar`, `OstatokStr`.

Упражнение 7. Изучите текст программы, содержащей один из вариантов реализации алгоритма «бегущая строка».

```
program DemoStringGo;    {Программа "бегущая строка"}
uses
  Crt;                   {Подключение специализированного модуля, содержащего стандартные
процедуры и функции управления дисплеем и клавиатурой компьютера}
type
  Stroka = string[160];
var
  Vhod: Stroka;
  procedure GoString (X,Y:byte; InSt:Stroka); {X,Y – координаты "бегущей строки
InSt – формальный параметр-значение типа Stroka для передачи из основной
программы текста сообщения, который должен "бежать" на экране}
  var
    St1: Stroka;
    I: byte;
  begin
    {Начало GoString}
    St1:= ' '; {Начальное значение строки St1}
    ClrScr;    {Очистить экран}
    St1:= St1+InSt; {Приклеивание к строке St1 значения InSt справа}
    for I:=1 to Length(St1) do
      begin
        Delete(St1,1,1); {Удаление 1 символа из строки St1, начиная с позиции 1
и смещение за счет этого строки St1 влево на 1 символ}
        GoToXY(X,Y); {Стандартная процедура специализированного модуля Crt,
предназначенная для перемещения курсора в позицию с координатами X, Y}
        Write(St1);
        Sound(1000); {Звуковое сопровождение движения строки}
        Delay(5);
        NoSound;
        Delay(90);
        DelLine {Стандартная процедура специализированного модуля Crt,
предназначенная для удаления строки, отмеченной курсором}
      end
    end;    {Конец GoString}
  begin {Начало основной программы DemoStringGo}
    GoString(1, 10, 'Для получения бумажной копии включите принтер!!!');
    {Вызов процедуры GoString с параметрами – значениями 1,10 – координатами начал
строки и фактическим параметром-значением – текстом сообщения}
    Vhod:= 'Установите бумагу !!!';
```

```
St1, 1, 14, Vhod)    {Вызов процедуры GoString с параметрами –
значениями 1, 14 – координатами начала строки и фактическим
параметром – значением Vhod – текстом сообщения}
{Начало основной программы DemoStringGo}
```

Как видно из текста программы, эффект «бегущей строки» создается путем циклического удаления из выводимой на экран строки заранее размещенных слева от нее пробелов.

Запустите интегрированную среду программирования. Введите текст программы `DemoStringGo` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции задайте для просмотра в окне отладчика величины `I`, `X`, `Y`, `Vhod`, `St1`. Сделайте видимыми одновременно окно редактора с текстом программы и окно просмотра. Выполните программу в пошаговом режиме с трассировкой процедуры `GoString` и проследите за изменениями значений переменных в основной программе и в подпрограмме-процедуре, обратите внимание на передачу параметров при вызове процедуры.

Сфера применения процедуры `GoString` может быть весьма широка: от вывода сопровождаемых звуковым сигналом предупредительных и аварийных сообщений до организации элементов меню.

Контрольные вопросы и задания

Вопросы

1. Что такое строка?
2. Каким идентификатором определяются данные строкового типа?
3. Какова максимально возможная длина строки? Как определить текущую длину строки?
4. Какие выражения называются строковыми?
5. Какие операции допустимы над строковыми данными?
6. Каким образом производится сравнение строк?
7. Какие требования предъявляются к записи выражений с операндами строкового и литерного типа?
8. Как можно обратиться к отдельным символам строки?
9. Объясните назначение специальных процедур и функций обработки данных строкового типа. Приведите примеры.

Задания

1. Напишите программу, подсчитывающую количество букв во введенном с клавиатуры слове. Ввод осуществляйте в цикле `while do`. Выход из программы – строка '999'.

2. Напишите программу, подсчитывающую количество вхождений заданной буквы в введенной строке.
3. Напишите программу, которая вводит строку и выводит ее, сокращая каждый раз на 1 символ до тех пор, пока в строке не останется 1 символ.
4. Напишите программу, определяющую число слов в строке. Одно слово другого отделяется 1 пробелом.
5. Напишите программу, определяющую, является ли введенное слово числом.
6. Введите 2 целых числа. Преобразуйте числа в две строки, объедините их одну строку и выведите на экран результат.
7. Напишите программу, которая удаляет из введенной строки любой введенный с клавиатуры символ. Процесс удаления выделите в отдельную процедуру DelChInString, строку и символ определите как глобальные переменные. В результате должна получиться программа-модель работы одного из режимов любого текстового редактора.
8. Напишите программу, удаляющую из введенной строки все пробелы. Для удаления реализуйте отдельную функцию NewStr и примените в ней оператор Repeat и функцию Pos.
9. Напишите программу, сортирующую символы введенной с клавиатуры строки в порядке возрастания их номеров в ASCII-таблице. Например, если введено: 'CBA', в результате надо получить 'ABC'.
10. Вычислите длину самого короткого слова в предложении из трех слов, разделенных пробелами.
11. Выясните, какая из букв — первая или последняя — встречается в заданном слове чаще.
12. Задано существительное первого склонения, оканчивающееся на «а». Напечатайте это слово во всех падежах.
13. Сколько букв «у» в слове стоит на четных местах?
14. Замените в заданном слове все буквы «о» пробелами.
15. В тексте, состоящем из латинских букв и заканчивающемся точкой, подсчитайте количество гласных букв.
16. Даны два слова. Поменяйте местами буквы этих слов, занимающие одинаковые позиции.
17. Заданы фамилия, имя и отчество учащегося, разделенные пробелом. Напечатайте его фамилию и инициалы.
18. Вычеркните i-ю букву слова.
19. Дан текст, в котором слова разделены пробелами.
 - 1) подсчитайте число слов в тексте;
 - 2) найдите самое длинное слово текста (длина текста 100 символов).

20. Задан текст, состоящий из слов, разделенных одним или несколькими пробелами. Сформируйте новый текст, включив в него слова заданного текста, разделенные только одним пробелом.
21. Сложное слово состоит из двух частей одинаковой длины и соединительной гласной. Найдите обе части этого слова.
22. Удалите из заданного слова все буквы, совпадающие с его последней буквой.
23. Удалите из слова X те буквы, которые встречаются в слове Z.
24. Подсчитайте число различных букв в слове.
25. Напишите программу подсчета числа вхождений в текст заданного фрагмента (цепочки символов). Например, в тексте «банан упал на барабан» фрагмент «ба» встречается 3 раза.
26. Напишите программу, которая по числу, меньшему 1000, написанному арабскими цифрами, формирует его название.
27. Напишите программу, которая по названию числа, меньшего 1000, написанному на русском (английском) языке, формирует его цифровую запись.
28. Даны два слова. Напишите программу, определяющую, можно или нет из букв слова A составить слово B.
29. Напишите программу перевода строки строчных русских букв в прописные.
30. Напишите программу, вычеркивающую каждую третью букву слова X.
31. Напишите программу подсчета числа одинаковых букв, стоящих на одних и тех же позициях в словах X и Y.
32. Напишите программу, выясняющую, на гласную или согласную букву оканчивается слово X.
33. Напишите программу вычисления суммы позиций, на которых в слове X стоят буквы «в» и «п».
34. Напишите программу шифрования текстового сообщения. Можно использовать следующий способ шифрования: шифровальщик задает ключ шифровки — целое число, который определяет величину смещения букв русского алфавита, например: ключ = 3, тогда в тексте буква «а» заменяется на «г» и т. д. Используются все буквы русского алфавита. Е считается дважды.
35. Напишите программу дешифрования текстового сообщения, зашифрованного программой из предыдущего задания.

Глава 7

Массивы

Рассмотренные ранее простые типы определяют различные множества атомарных (неразделимых) значений. В отличие от них, структурные типы задают множества сложных значений, каждое из которых образует совокупность нескольких значений другого типа. В структурных типах выделяют регулярный тип (массивы).

Описание типа «массив»

С понятием «массив» приходится сталкиваться при решении научно-технических и экономических задач обработки совокупностей большого количества значений. В общем случае массив — это структурированный тип данных состоящий из фиксированного числа элементов, имеющих один и тот же тип. Название *регулярный тип* (или *ряд*) массивы получили за то, что они объединяют однотипные (логически однородные) элементы, упорядоченные по индексам, определяющим положение каждого элемента в массиве.

В качестве элементов массива можно использовать любой другой ранее описанный тип, поэтому вполне правомерно существование массивов записей, массивов указателей, массивов строк, массивов массивов и т. д. Элементами массива могут быть данные любого типа, включая структурированные. Тип элементов массива называется *базовым*. Особенностью языка Pascal является то, что число элементов массива фиксируется при описании и в процессе выполнения программы не меняется.

ПРИМЕЧАНИЕ

Начиная с Delphi 5 в Object Pascal введены динамические массивы. Они отличаются от обычных статических массивов тем, что в них не объявляется заранее длина — число элементов.

Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует совокупность номеров (индексов), определяющих его местоположение в общей последовательности. Доступ к каждому отдельному элементу осуществляется путем индексирования элементов массива. Индексы представляют собой выражения любого скалярного типа, кро-

вещественного. Тип индекса определяет границы изменения значений индекса. Для описания массива предназначено словосочетание *array of* (массив из). Синтаксическая диаграмма регулярных типов имеет следующий вид:



Исходя из синтаксической диаграммы, формат записи будет таким:

```
type  
  <имя т. ...> = array[тип индекса] of <тип компонента>;  
var  
  <идентификатор...> : <имя типа>;
```

Массив может быть описан и без представления типа в разделе описания типов данных:

```
var  
  <идентификатор...> . array[тип индекса] of <тип компонента>;
```

Пример:

```
type  
  Klass = (K1, K2, K3, K4);  
  Znak = array[1..255] of char;  
var  
  M1: Znak; {Тип Znak предварительно описан в разделе типов}  
  M2: array[1..60] of integer; {Прямое описание массива M2}  
  M3: array[1..4] of Klass;  
  Mas: array[1..4] of integer;
```

Если в качестве базового типа взят другой массив, то образуется структура, которую принято называть *многомерным массивом*.

Пример:

```
type  
  Vector = array[1..4] of integer;  
  Matrix = array[1..4] of Vector;  
var  
  Matr: Matrix;
```

Ту же структуру можно получить, используя другую форму записи:

```
var  
  Matr: array[1..4, 1..4] of integer;
```

Если при такой форме описания массива задан один индекс, массив называется *одномерным*, если два индекса — *двумерным*, если *n* индексов — *n-мерным*. Одномерный массив соответствует понятию линейной таблицы (вектора), двумерный — понятию прямоугольной таблицы (матрицы, набора векторов).

Размерность ограничена только объемом памяти конкретного компьютера. Одномерные массивы обычно используются для представления векторов, а двумерные — для представления матриц.

Пример:

```
var
  VectorZ: array[1..40] of real; {Одномерный массив из 40 элементов}
  MatrU : array[1..8,1..8] of byte; {Двумерный массив 8x8 элементов}
  Trilf : array[1..4,1..5,1..8] of integer; {Трёхмерный массив}
```

Для описания массива можно использовать предварительно определенные константы:

```
const
  G1 = 4; G2 = 6;
var
  MasY : array[1..G1,1..G2] of real;
```

Элементы массива располагаются в памяти последовательно. Элементы с меньшими значениями индекса хранятся в младших адресах памяти. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает первым.

Например, если имеется массив `A:array[1..5,1..5] of integer`; то в памяти элементы массива будут размещены по возрастанию адресов:

```
A[1,1]
A[1,2]
...
A[1,5]
A[2,1]
A[2,2]
...
A[5,5]
```

Контроль правильности значений индексов массива может проводиться с помощью директивы компилятора R. По умолчанию директива R находится в пассивном состоянии `{R-}`. Перевод в активное состояние вызывает проверку всех индексных выражений на соответствие их значений диапазону типа индекса.

Существует различие между регулярными типами в языке Pascal и массивами в некоторых других языках программирования, заключающееся в том, что в Pascal количество элементов массива всегда должно быть фиксированным, т. е. определяться при трансляции программы. Это считается недостатком языка, так как не во всех программах можно заранее предсказать необходимый размер массива (который может определяться в зависимости от типа или иных условий, возникающих в процессе исполнения). В программах, обрабатывающих массивы, помимо использования для определения размера массива предварительно определенных констант иногда используется прием

позволяющий имитировать работу с массивами переменной длины, который заключается в следующем: в разделе описания констант предварительно определяют возможное максимальное значение размера массива, а затем в программе запрашивают текущее значение размера и используют это значение далее при заполнении и обработке массива.

Операции над массивами

Для работы с массивом как единым целым используется идентификатор массива без указания индекса в квадратных скобках. Массив может участвовать только в операциях отношения «равно», «не равно» и в операторе присваивания. Массивы, задействованные в этих операциях, должны быть идентичны по структуре, т. е. иметь одинаковые типы индексов и одинаковые типы компонентов. Например, если массивы A и B описаны как

```
var
  A, B : array [1..20] of real;
```

то применение к ним допустимых операций даст следующий результат:

Выражение	Результат
<code>A=B</code>	True, если значение каждого элемента массива A равно соответствующему значению элемента массива B
<code>A<>B</code>	True, если хотя бы одно значение элемента массива A не равно значению соответствующего элемента массива B
<code>A:=B</code>	Все значения элементов массива B присваиваются соответствующим элементам массива A. Значения элементов массива B остаются неизменными

Операции над элементами массива

После объявления массива каждый его элемент можно обработать, указав идентификатор (имя) массива, а за ним — индекс элемента в квадратных скобках. Например, запись `Mas[2], VectorZ[10]` позволяет обратиться ко второму элементу массива Mas и десятому элементу массива VectorZ. При работе с двумерным массивом указываются два индекса, с n-мерным массивом — n индексов. Например, запись `MatrU[4,4]` делает доступным для обработки значение элемента, находящегося в четвертой строке четвертого столбца массива MatrU.

Индексированные элементы массива называются *индексированными переменными* и могут использоваться так же, как и простые переменные. Например, они могут входить в выражения в качестве операндов, использоваться в операторах `for`, `while`, `repeat`, входить в качестве параметров в операторы `Read`, `Readln`, `Write`, `Writeln`; им можно присваивать любые значения, соответствующие их типу.

Рассмотрим типичные ситуации, возникающие при работе с данными типа array. Для этого опишем три массива и четыре вспомогательные переменные:

```
var
  A, D      : array[1..4] of real;
  B         : array[1..10, 1..15] of integer;
  I, J, K   : integer;
  S         : real;
```

Инициализация (присваивание начальных значений) массива заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу. Наиболее эффективно эта операция выполняется с помощью оператора for, например: for I := 1 to 4 do A[I] := 0;

Для инициализации двумерного массива обычно используется вложенным оператор for, например:

```
for I := 1 to 10 do
  for J := 1 to 15 do
    B[I, J] := 0;
```

Pascal не имеет средств ввода-вывода сразу всех элементов массива, поэтому ввод и вывод значений производится поэлементно. Значения элементам массива можно присвоить с помощью оператора присваивания, как показано в примере инициализации, однако чаще всего они вводятся с клавиатуры при помощи оператора Read или Readln с использованием оператора цикла for:

```
for I:=1 to 4 do Readln(A[I]);
```

Аналогично, значения двумерного массива вводятся с помощью вложенного оператора for:

```
for I := 1 to 10 do
  for J := 1 to 15 do
    Readln (B[I, J]);
```

В связи с тем что использовался оператор Readln, каждое значение будет вводиться с новой строки. Можно ввести значения отдельных элементов а не всего массива. Так, при помощи операторов Read(A[3]); Read(B[6,9]); вводятся значения третьего элемента вектора A и значения элемента, расположенного в шестой строке девятого столбца матрицы B. Оба значения набираются в одной строке экрана, начиная с текущей позиции курсора.

Вывод значений элементов массива выполняется аналогичным образом, используются операторы Write или Writeln:

```
for I := 1 to 4 do
  Writeln (A[I]);           {Вывод значений массива A}
```

или

```
for I := 1 to 10 do
  for J := 1 to 15 do
    Writeln (B[I, J]);      {Вывод значений массива B}
```

Копированием массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Копирование можно выполнить при помощи одного оператора присваивания, например A:= D; или с помощью оператора for:

```
A[I] := D[I];
```

В обоих случаях значения элементов массива D не изменяются, а значения элементов массива A становятся равными значениям соответствующих элементов массива D. Очевидно, что оба массива должны быть идентичны по структуре.

Иногда требуется осуществить поиск в массиве каких-либо элементов, удовлетворяющих некоторым известным условиям. Пусть, например, требуется выяснить, сколько элементов массива A имеют нулевое значение. Для ответа на этот вопрос введем дополнительную переменную K и воспользуемся операторами for и if:

```
K := 0;
for I := 1 to 4 do
  if A[I] = 0 then K := K + 1;
```

После выполнения цикла переменная K будет содержать число элементов массива A, имеющих нулевое значение.

Перестановка значений элементов массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массива. Например, фрагмент программы, обменивающий значения первого и пятого элементов массива A, будет выглядеть следующим образом:

```
Vs := A[5];           {Vs — вспомогательная переменная}
A[5] := A[1];
A[1] := Vs;
```

Упражнение 1. Создадим программу, которая формирует одномерный массив тремя способами (по формуле, случайным образом, вводом элементов с клавиатуры) и выводит полученный в каждом случае массив на экран.

В начале программы опишем переменные: массив Massiv из десяти вещественных чисел и целую переменную I, которая будет выполнять функции параметра цикла и использоваться для указания номера (индекса), определяющего местоположение элемента в массиве. Описание переменных будет представлено следующим образом:

```
var
  Massiv : array [1..10] of real;
  I : integer;
```

Основная часть программы будет состоять из трех отдельных частей, каждая из которых иллюстрирует разный способ формирования одномерного массива. Первый фрагмент — формирование одномерного массива по формуле.

Так как инициализация массива для всех его элементов выполняется одинаковым образом, т. е. является повторяющейся операцией и число повторений известно заранее, то она может быть записана оператором цикла с параметром. Если каждый элемент массива равен утроенному значению его порядкового номера (индекса), то этот фрагмент программы можно записать так:

```
for I:=1 to 10 do
  Massiv[I]:= I*3;
```

Во второй части программы выполним формирование одномерного массива, задавая значение каждого элемента как результат вычисления случайной функции Random(I). Для использования этой функции подключим к программе стандартный модуль Crt при помощи команды uses Crt. Эта часть программы также выполняется циклически, поэтому инициализацию массива осуществим в цикле for, в теле которого будет выполняться вычисление случайного числа при помощи функции Random(Count), после чего это значение будет присваиваться очередному, I-му элементу массива.

```
Randomize;
for I:=1 to 10 do
  Massiv[I]:=Random(I);
```

В третьей части программы формирование одномерного массива выполняется в результате повторяющегося ввода с клавиатуры значений каждого элемента, что может быть реализовано следующим образом:

```
for I:=1 to 10 do
begin
  Write('Введите ',I,'-й элемент массива ');
  Readln(Massiv[I]);
end;
```

Для просмотра значений созданного массива необходимо выводить их на экран. Поскольку это требуется делать три раза, то имеет смысл оформить вывод значений элементов массива в виде отдельной процедуры, которая полчается из точки вызова основной программы фактическое значение локальной переменной Msg — сообщение о том, каким способом сформированы значения элементов массива, печатает его на экране и выводит значения всех элементов массива. Текст процедуры может быть записан так:

```
procedure Out_Massiv(Msg:string); {Процедура вывода массива на экран}
var J : integer;
begin
  Writeln(Msg);
  for J:=1 to 10 do
    Writeln(J,'-й элемент массива =',Massiv[J]:5);
  Writeln;
end;
```

Для вызова процедуры после каждой самостоятельной части программы запишем ее имя, а в качестве фактического параметра-значения укажем текст сообщения, выводимого перед печатью значений массива, например:

```
Out_Massiv('Одномерный массив, элементы которого = I*3');
```

В целом текст программы будет выглядеть следующим образом:

```
var
  Massiv : array [1..10] of real;
  integer;

procedure Out_Massiv(Msg:string); {Процедура вывода массива на экран}
var J : integer;
begin
  Writeln(Msg);
  for J:=1 to 10 do
    Writeln(J,'-й элемент массива =',Massiv[J]:5);
  Writeln;
end;

begin {Основная программа}
  for I:=1 to 10 do {Формирование одномерного массива по формуле}
    Massiv[I] := I*3;
  {Вызов процедуры вывода массива на экран с передачей ей параметра-значения
  текстового сообщения}
  Out_Massiv('Одномерный массив, элементы которого =I*3');
  Randomize; {Формирование одномерного массива случайной функцией Random(I)}
  for I:=1 to 10 do
    Massiv[I]:=Random(I);
  Out_Massiv('Одномерный массив случайных значений функции Random(20)');
  for I:=1 to 10 do {Формирование одномерного массива вводом с клавиатуры}
  begin
    Write('Введите ',I,'-й элемент массива ');
    Readln(Massiv[I]);
  end;
  Out_Massiv('Одномерный массив введенный с клавиатуры');
```

Запустите интегрированную среду программирования. Введите текст программы Demo_massiv1 и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы.

Упражнение 2. Создадим программу, формирующую одномерный массив путем ввода с клавиатуры, находящую в массиве элементы, заданные пользователем, подсчитывает их количество и выводящую номер первого найденного элемента.

В разделе описания констант укажем значение константы Count = 30. В программе значение этой константы определяет количество элементов массива.

Использование константы в описании размеров массива предпочтительнее, так как в случае изменения размеров массива не придется вносить изменения в весь текст программы, а достаточно будет только один раз указать в разделе описания констант новое значение константы Count. В связи с этим описание массива зададим так:

```
M : array [1..Count] of Byte;
```

Введем переменные целого типа: N — значение искомого элемента; A — номер первого элемента массива, значение которого равно N; B — количество таких элементов в массиве; I — переменная, выполняющая функции параметра цикла и одновременно служащая указателем номера очередного элемента массива. Массив элементов опишем в разделе констант как упорядоченный набор целых данных типа byte (2,2,3,4,5,2,7,8,9,2). Тогда раздел описания констант и переменных в программе будет таким:

```
const
  Count = 10;
  M : array [1..Count] of byte=(2,2,3,4,5,2,7,8,9,2);
var
  N, A, B, I : Byte;
```

В начале программы распечатаем значения элементов массива:

```
WriteLn('Исходный массив:');
for I := 1 to Count do Write(M[I]:3, ' '); WriteLn; WriteLn;
```

В связи с тем, что ни один подходящий элемент еще не найден, присвоим переменным A и B значение 0. Затем выведем на экран приглашение на ввод значения искомого элемента и считаем это значение с клавиатуры. На Pascal эти действия будут записаны следующим образом:

```
Write('Введите значение элемента массива для поиска >');
ReadLn(N);
```

Поиск элемента массива, значение которого равно введенному числу N, выполняется путем циклического сравнения значений всех элементов от первого до последнего со значением числа N, поэтому мы запишем его в виде цикла с параметром. Оператор if M[I] = N then выполняет сравнение значения очередного элемента массива с заданным значением N. Если условие M[I] = N выполняется, то счетчик числа найденных элементов B увеличивается на единицу. Так как нам требуется найти номер первого элемента, т. е. при первом выполнении условия M[I] = N запомнить номер данного элемента, то можно записать:

```
if B = 0 then A := I.
```

Тогда блок поиска нужного элемента можно будет записать так:

```
for I := 1 to Count do
  if M[I] = N then
    begin
```

```
    if B = 0 then A := I;
    B := B + 1;
  end;
```

В заключительной части программы на экран должно выводиться сообщение о том, что в массиве нет искомого элемента, или сообщение о количестве элементов массива, имеющих значение, равное N, и печататься номер первого такого элемента. Это можно записать следующим образом:

```
if B=0 then WriteLn('Нет таких элементов в массиве')
else
  begin
    WriteLn('Количество эл-тов массива, имеющих значение '.N.' - '. B);
    WriteLn('Первый элемент - '. A);
  end;
```

В целом текст программы будет таким:

```
program Find_Elem; {Поиск элемента в массиве}
const
  Count = 10;
  M : array [1..Count] of byte=(2,2,3,4,5,2,7,8,9,2);
var
  N, A, B, I : Byte;
begin
  WriteLn('Исходный массив:');
  for I := 1 to Count do Write(M[I]:3, ' '); WriteLn; WriteLn;
  A := 0; {Нет элемента с таким значением}
  B := 0; {Пока не найдено ни одного элемента}
  Write('Введите значение элемента массива для поиска >');
  ReadLn(N);
  for I := 1 to Count do {Поиск элемента, значение которого =N}
    if M[I] = N then
      begin
        if B = 0 then A := I; {Запомнить номер первого элемента, равного N}
        B := B + 1; {Увеличить число найденных элементов на 1}
      end;
    if B=0 then WriteLn('Нет таких элементов в массиве')
    else
      begin
        WriteLn('Количество элементов массива, имеющих значение '.N.' - '. B);
        WriteLn('Первый элемент - '. A);
      end;
  end;
```

Запустите интегрированную среду программирования. Введите текст программы Find_Elem и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проверьте работу программы, задавая различные значения искомого элемента, например: 2,5,9.

Сортировка массивов

Сортировка — одна из наиболее распространенных операций обработки данных. Сортировкой называется распределение элементов множества по группам в соответствии с определенными правилами. Например, сортировка элементов массива, в результате которой получается массив, каждый элемент которого, начиная со второго, не превосходит элемент, стоящий от него слева, называется *сортировкой по невозрастанию*.

Упражнение 3. Создадим программу, которая выводит на экран несортированный массив целых чисел, затем выполняет его сортировку по невозрастанию и выводит на экран отсортированный массив.

Рассмотрим три метода сортировки одного и того же массива M из 20 целых чисел. Использование одного и того же массива позволит объективно сравнить эффективность разных методов. В связи с этим описание массива во всех примерах выполним следующим образом:

```
const
  Count = 20;
  M:array [1..Count] of byte=(9,11,12,3,19,1,5,17,10,18,3,19,
17,9,12,20,20,19,2,5);
```

Для сравнения эффективности разных способов сортировки введем целую переменную A , значение которой будет равно числу итераций (повторов просмотра массива). Для наблюдения текущего состояния массива после каждой перестановки элементов будем выводить его на экран:

```
for L := 1 to Count do Write(' ', M[L]); Writeln('Число итераций = ', A);
```

Линейная сортировка (сортировка отбором)

Идея линейной сортировки по невозрастанию заключается в том, чтобы последовательно просматривая весь массив, отыскать наибольшее число и поместить его на первую позицию, обменяв его с элементом, который ранее занимал первую позицию. Затем просматриваются все остальные элементы массива и выполняется аналогичная операция по отбору из рассматриваемой части массива максимального элемента и обмену местами этого элемента и первого в рассматриваемой части, и т. д.

Введем в разделе описания следующие целые переменные:

I — для указания позиции первого элемента в рассматриваемой части массива;

J — для указания позиции очередного сравниваемого с I -м элемента;

N — для временного хранения значения первого элемента для обмена значениями с максимальным из рассматриваемой части массива;

L — параметр цикла при выводе текущего значения элементов массива в процессе сортировки для отслеживания происходящих в массиве изменений;

A — переменная, значение которой будет равно числу перестановок элементов.

Вывод исходного массива на экран запишем следующим образом:

```
Writeln('Исходный массив:');
for I := 1 to Count do Write(' ', M[I]); Writeln;
```

Перед началом сортировки установим значение счетчика итераций A равным 0. Для сортировки организуем два цикла **for**: внешний цикл с параметром I , указывающим позицию первого элемента в неотсортированной части массива, и внутренний цикл с параметром J , указывающим позицию очередного, сравниваемого с первым, элемента неотсортированной части массива. Сравнение элементов запишем при помощи оператора:

```
if M[I] > M[J] then
begin
  N := M[I];
  M[I] := M[J];
  M[J] := N;
end;
```

Если условие $M[I] < M[J]$ выполняется, т. е. в неотсортированной части массива найден элемент, больший первого, то следует поменять местами эти элементы. Перестановка осуществляется следующим образом: сначала значение $M[I]$ запоминается в переменной N , затем значение элемента массива из J -й позиции записывается в I -ю позицию, а после этого в J -ю позицию массива записывается значение переменной N . Для отслеживания изменений состояния массива после каждой перестановки зададим цикл вывода значений всех элементов массива и напечатаем текущее значение числа перестановок элементов A :

```
for L := 1 to Count do Write(' ', M[L]); Writeln('Число итераций = ', A);
```

Полный текст программы линейной сортировки массива M по невозрастанию может быть записан следующим образом:

```
Program Sort_Linear; {Линейная сортировка по невозрастанию}
const
  Count = 20;
  M:array [1..Count] of byte=(9,11,12,3,19,1,5,17,10,18,3,19,17,9,12,20,20,19,2,5);
var
  I, J, N: byte;
  A: integer;
begin
  Writeln('Исходный массив:');
  for I := 1 to Count do Write(' ', M[I]); Writeln;
  A:=0;
  for I := 1 to Count - 1 do {Изменяем размер неотсортированной части массива}
  for J:=I+1 to Count do {Сравниваем поочередно I-й элемент неотсортированной части массива со всеми от I+1-го до конца}
  begin
```



```

A=A+1;
M[I] < M[J] then {Если в несортированной части массива нашли элемент
льший чем I-й, то поменять их местами}
begin
  N := M[I];           {Запомнить на время значение M[I]}
  M[I] := M[J];
  M[J] := N;
end;
L := 1 to Count do Write(' ', M[L]); WriteLn('Число итераций =', A);

```

Запустите интегрированную среду программирования. Введите текст программы Sort_Lin и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции запустите программу, отслеживая в пошаговом режиме текущие значения переменных: I, J, N, M[I], M[J], а также просматривая на экране пользователя результат изменения массива за один проход внутреннего цикла. По последнему значению A можно определить, что для данного массива линейная сортировка по невозрастанию выполняется за 190 итераций.

Сортировка методом пузырька

Один из самых популярных методов сортировки — «пузырьковый» метод — основан на том, что в процессе исполнения алгоритма более «легкие» элементы массива постепенно «всплывают». Особенностью данного метода является сравнение не каждого элемента со всеми, а сравнение в парах соседних элементов. Алгоритм пузырьковой сортировки по убыванию состоит в последовательном просмотре снизу вверх (от начала к концу) массива M. Если для соседних элементов выполняется условие, согласно которому элемент, находящийся справа, больше элемента, находящегося слева, то выполняется обмен значениями этих элементов.

Текст программы сортировки массива по невозрастанию можно записать следующим образом:

```

Sort_Puz: {Сортировка массива «пузырьковым» методом по невозрастанию}

  Count := 20;
  array [1..Count] of byte = (9, 11, 12, 3, 19, 1, 5, 17, 10, 18, 3, 19, 17, 9, 12, 20, 20, 19, 2, 5);
  var
    K, L: Byte;
    A: Integer;
  begin
    WriteLn('Исходный массив:');
    for I := 1 to Count do Write(M[I], ' '); WriteLn;
    A := 0;
    for I := 2 to Count do {Сортировка «пузырьковым» методом по невозрастанию}

```

```

begin
  for J := Count downto 1 do
    begin
      A := 0;
      if M[J-1] < M[J] then {Если элемент справа больше элемента слева,
то «вытеснить» его влево — пузырек «всплывает»}
      begin
        M[J-1] := M[J]; {Обмен значениями этих элементов}
        M[J] := M[J-1];
        M[J-1] := M[J];
        M[J] := K;
      end;
      {Печатать текущее состояние массива после каждой перестановки}
      for L := 1 to Count do Write(' ', M[L]); WriteLn('Число итераций =', A);
    end;
  end;
end. {Завершение сортировки}
end.

```

Запустите интегрированную среду программирования. Введите текст программы Sort_Puz и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции запустите программу, отслеживая в пошаговом режиме текущие значения переменных I, J, K, M[J], M[J-1], всего массива M, а по окончании цикла J просматривая изменения массива в результате одного прохода. Как можно заметить, при помощи оператора for с параметром J выполняется циклический просмотр элементов массива от последнего до второго, и при этом элементы, большие, чем «соседи» слева, смещаются при обмене к началу массива, а меньшие смещаются вправо — «всплывают» в конце массива. Каждый проход фиксируется счетчиком — увеличением на единицу значения переменной A. После каждого просмотра-сравнения элементов массива просматриваемый участок массива уменьшается на один элемент, так как из рассмотрения исключается самый левый (самый большой) элемент. Такое уменьшение просматриваемого участка реализуется при помощи внешнего цикла for с параметром I. По последнему значению A можно определить, что для данного массива сортировка по невозрастанию пузырьковым методом выполняется за 170 итераций.

Метод быстрой сортировки с разделением

Оба рассмотренных выше метода достаточно просты и наглядны, но не очень эффективны. Значительно быстрее работает алгоритм сортировки К. Хоора, который называют *сортировкой с разделением*, или *быстрой сортировкой*. В основу алгоритма положен метод последовательного деления массива на части. Рассмотрим пример программы, реализующей этот метод.

```

program Quick_sort; {Быстрая сортировка по невозрастанию дроблением массива на части}
uses Crt;
const Count=20;

```

```

M array [1..Count] of byte=(9,11,12,3,19,1,5,17,10,18,3,19,17,9,12,20,20,19,2
var
  I,A integer.
procedure QuickS(First,Last:integer);
  var I,J,X,W,L:integer;
begin
  I:=First;      {Левая граница массива – первый элемент}
  J:=Last;       {Правая граница массива – последний элемент}
  X:=M[(First+Last) div 2]; {Определить середину массива}
  repeat
    while M[I]>X do I:=I+1;
    while X>M[J] do J:=J-1;
    A:=A+1;      {Увеличить число итераций обмена элементов}
    if I<=J then
      begin      {Поменять местами элементы}
        W:=M[I];
        M[I]:=M[J];
        M[J]:=W;
        I:=I+1;
        J:=J-1;
      end;
    {Печатать текущее состояние массива после каждой перестановки}
    for L := 1 to Count do Write(' ',M[L]); WriteLn('Число итераций ='..A)
  end;
  until I>J;
  if First<J then QuickS(First,J); {Рекурсивный вызов процедуры QuickS}
  if I<Last then QuickS(I,Last);  {Рекурсивный вызов процедуры QuickS}
end;      {Конец сортировки}
begin      {Начало основной программы}
  ClrScr;
  WriteLn('Исходный массив:'); WriteLn;
  for I:=1 to Count do Write (M[I]:3..' '); WriteLn;
  A:=0;
  QuickS (1,Count);      {Вызов процедуры сортировки QuickS}
  WriteLn ('Отсортированный массив:'); WriteLn;
  for I:=1 to Count do Write (M[I]:3..' '); WriteLn;
end.

```

Запустите интегрированную среду программирования. Введите текст программы Quick_sort и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции нажатие клавиши F7 запустит программу в пошаговом режиме с трассировкой процедуры QuickS и проследите за изменениями текущих значений переменных I, J, M[I], M[J], First, X, Last. В момент вызова процедуры просматривайте экран пользователя с текущим состоянием сортируемого массива или задавайте в окне просмотра имя массива M для наблюдения за изменением состояния массива в процессе сортировки.

После первого вызова процедуры QuickS в исходном массиве:

9 11 12 5 17 10 18 3 19 17 9 12 20 20 19 2 5

будет определена середина (10-й элемент) и переменной X присвоено значение M[10], т. е. 18. После этого массив делится на две части:

9 11 12 5 17 10 18 и 3 19 17 9 12 20 20 9 2 5

Далее выполняется перестановка элементов по следующему правилу:

При просмотре левой части массива слева направо выполняется поиск такого элемента массива, что $M[I] > X$, затем при просмотре правой части справа налево отыскивается такой элемент, что $M[J] < X$. Выполняется перестановка этих элементов, и так до тех пор, пока все элементы слева от середины, удовлетворяющие условию $M[I] > X$, не поменяются местами с элементами, расположенными справа от середины и удовлетворяющими условию $M[J] < X$. В результате получится массив из двух частей следующего вида:

9 11 12 3 19 1 5 17 10 18 3 19 17 9 12 20 20 19 2 5 Число итераций = 1
 19 20 12 3 19 1 5 17 10 18 3 19 17 9 12 20 11 9 2 5 Число итераций = 2
 19 20 20 3 19 1 5 17 10 18 3 19 17 9 12 12 11 9 2 5 Число итераций = 3
 19 20 20 19 19 1 5 17 10 18 3 3 17 9 12 12 11 9 2 5 Число итераций = 4
 19 20 20 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 5

Далее рекурсивно левая часть, в свою очередь, делится на две части, и вызывается процедура для сортировки левой части (с 1-го по 6-й элемент):

20 20 19 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 7
 20 20 19 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 8

После того как левая часть массива будет отсортирована, опять рекурсивно вызывается процедура, в которой определяется середина данной части массива, и выполняется перестановка элементов. Массив становится таким:

20 20 19 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 9
 20 20 19 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 10

Далее опять рекурсивно вызывается процедура сортировки, пока в каждой из частей не останется по одному элементу. Затем рекурсивно вызывается процедура для аналогичной сортировки правой части (с 7-го по 13-й элемент). Результат последовательных этапов сортировки массива выглядит так:

20 20 19 19 19 18 5 17 10 1 3 3 17 9 12 12 11 9 2 5 Число итераций = 11
 20 20 19 19 19 18 17 17 10 1 3 3 5 9 12 12 11 9 2 5 Число итераций = 12
 20 20 19 19 19 18 17 17 10 1 3 3 5 9 12 12 11 9 2 5 Число итераций = 13
 20 20 19 19 9 18 17 17 10 9 3 3 5 9 12 12 11 1 2 5 Число итераций = 14
 20 20 19 19 19 18 17 17 10 9 11 3 5 9 12 12 3 1 2 5 Число итераций = 15
 20 20 19 19 19 18 17 17 10 9 11 12 5 9 12 3 3 1 2 5 Число итераций = 16
 20 20 19 19 19 18 17 17 10 9 11 12 12 9 5 3 3 1 2 5 Число итераций = 17
 20 20 19 19 19 18 17 17 10 9 11 12 12 9 5 3 3 1 2 5 Число итераций = 18

20 20 19 19 19 18 17 17 12 9 11 12 10 9 5 3 3 1 2 5 Число итераций = 19
 20 20 19 19 19 18 17 17 12 12 11 9 10 9 5 3 3 1 2 5 Число итераций = 20
 20 20 19 19 19 18 17 17 12 12 11 9 10 9 5 3 3 1 2 5 Число итераций = 21
 20 20 19 19 19 18 17 17 12 12 11 9 10 9 5 3 3 1 2 5 Число итераций = 22
 20 20 19 19 19 18 17 17 12 12 11 10 9 9 5 3 3 1 2 5 Число итераций = 23
 20 20 19 19 19 18 17 17 12 12 11 10 9 9 5 5 3 1 2 3 Число итераций = 24
 20 20 19 19 19 18 17 17 12 12 11 10 9 9 5 5 3 1 2 3 Число итераций = 25
 20 20 19 19 19 18 17 17 12 12 11 10 9 9 5 5 3 1 2 3 Число итераций = 26
 20 20 19 19 19 18 17 17 12 12 11 10 9 9 5 5 3 3 2 1 Число итераций = 27

По последнему значению A можно определить, что для данного массива сортировка по невозрастанию выполняется за 27 итераций.

Как видно из приведенных примеров, алгоритм «быстрой» сортировки дает лучшие результаты, чем пузырьковый метод, однако следует учесть, что в некоторых случаях это преимущество будет выражено менее ярко. Например, если применить эту программу для сортировки массива M, элементы которого имеют значения 29, 27, 24, 3, 23, 19, 19, 18, 1, 17, 15, 13, 1, 0, 9, 9, 8, 6, 6, 5, то сортировка будет выполнена за 28 итераций, т. е. время процесса и число итераций даже увеличатся, несмотря на то, что исходный массив в большей степени упорядочен, в то время как сортировка линейным методом состоит из 190 итераций для обоих массивов, а пузырьковым — 166 (для первого массива — 170). Измените строку с описанием массива на:

```
M:array[1..Count] of byte=(29,27,24,3,23,19,19,18,1,17,15,13,1,0,9,9,8,6,6,
```

и проверьте действие программ сортировки всеми рассмотренными методами на примере нового массива.

Бинарный поиск в упорядоченных массивах

Выше мы продемонстрировали использование метода линейного (последовательного) поиска, суть которого заключается в последовательном просмотре всего списка элементов массива и сравнении значения очередного элемента с заданным для поиска. Этот метод поиска вполне эффективен на неупорядоченных массивах данных, а на предварительно отсортированных массивах значительно эффективнее выполнять поиск другими методами. Например, метод линейного поиска практически бесполезен при поиске информации в массивах большого размера, так как занимает много времени.

Одним из эффективных методов поиска в больших отсортированных массивах является бинарный поиск, или поиск методом деления пополам. В основе этого метода лежит идея целенаправленных последовательных догадок о предполагаемом местоположении искомого элемента. Такой метод можно продемонстрировать на примере шуточного совета, как поймать в Африке. «Для начала разделим Африку пополам. Ясно, что лев находится только в одной половине. Та половина, в которой находится лев, снова делится пополам

и т. д. Площадь Африки приблизительно 30 млн км². Следовательно, после примерно 45 делений мы получим площадку, не превышающую 1 м². Теперь на такой площадке льва поймать нетрудно».

Аналогично выполняется бинарный поиск элемента в упорядоченном массиве. Вместо просмотра подряд всех элементов массива разделим его пополам. Так как массив отсортирован, то, сравнивая искомый элемент со значением среднего элемента массива, мы в состоянии сделать вывод о том, может ли элемент с таким значением присутствовать в массиве и в какой половине он находится, т. е. определить область дальнейшего поиска. Затем делится пополам та часть массива, в которой находится элемент, и так до тех пор, пока рассматриваемая часть массива получится состоящей из одного элемента.

Упражнение 4. Создадим программу, выполняющую бинарный поиск заданного элемента в отсортированном массиве целых чисел, элементы которого имеют значения: 20, 20, 19, 19, 19, 18, 17, 17, 12, 12, 11, 10, 9, 9, 5, 5, 3, 3, 2, 1.

Введем в разделе описания констант размер массива Count = 20 и опишем массив целых чисел:

```
M:array[1..Count] of byte=(20,20,19,19,19,18,17,17,12,12,11,10,9,9,5,5,3,3,2,1).
```

В разделе описания переменных опишем следующие переменные:

N — для хранения значения искомого элемента;

I — для указания очередного элемента массива;

First — указание индекса элемента, являющегося левой границей рассматриваемой части массива;

Last — указание индекса элемента, являющегося правой границей рассматриваемой части массива;

Found — логическая переменная, в которую будет записываться результат поиска;

A — переменная, значение которой будет равно числу перестановок элементов.

В начале программы реализуем вывод исходного массива на экран, затем вывод запроса искомого элемента, после чего считывание его значения в переменную N.

Перед началом поиска обнулим счетчик повторений A, установим левую и правую границы рассматриваемой части массива, а переменной Found присвоим значение False — элемент пока не найден. Данный фрагмент программы будет выглядеть так:

```
A:=0;  
First:=1;  
Last:=Count;  
Found:=False;
```

Повторяющуюся процедуру бинарного поиска реализуем с помощью оператора повтора repeat. Первый шаг бинарного поиска — деление исходного массива на две части — можно записать так:

```
I := (First + Last) div 2.
```

Условие поиска:

```
if M[I] = N then Found:=True
```

Если условие $M[I] = N$ выполняется, то искомый элемент найден, переменной Found присваивается значение True, и управление передается за пределы оператора if. Если условие не выполняется, то при помощи оператора

```
if M[I] > N then First := I+1
else Last := I-1;
```

проверяется условие $M[I] > N$. Если оно выполняется, значит значение искомого элемента расположено в правой части массива, поэтому левую границу части массива следует перенести в позицию $I+1$ ($First:=I+1$). Если условие не выполняется, то элемент нужно искать в левой части массива, а следовательно, правую границу части массива следует перенести в позицию $I-1$ ($Last:=I-1$). Затем нужно увеличить счетчик повторений при поиске $A:=A+1$. Оператор повтора завершим записью условия окончания (Found) or ($First>Last$). Если это условие выполняется, т. е. если найдется искомый элемент или будет просмотрен весь массив, цикл завершится. Этот фрагмент программы можно записать так:

```
repeat                                {Повторять поиск}
  I := (First + Last) div 2;          {Разделить на две части}
  if M[I] = N then Found:=True
  else
    begin
      if M[I] > N then First := I+1 {Искать элемент в правой части}
      else Last := I-1;             {Искать элемент в левой части}
    end;
  A:=A+1;                             {Увеличить счетчик числа итераций}
until (Found) or (First>Last); {Завершить, если найдется искомый
                                элемент или будет просмотрен весь массив}
```

В заключительной части программы запишем условие вывода результата поиска. Если значение переменной $Found = True$, то выведем на экран сообщение о том, какую позицию в массиве занимает искомый элемент, в противном случае выведем сообщение о том, что такого элемента в массиве нет. После чего выведем сообщение о том, сколько просмотров массива было выполнено в процессе поиска.

Текст программы будет выглядеть следующим образом:

```
program Find_Bin;
const
  Count = 20;
```

```
M : array[1..Count] of byte =
(20,20,19,19,19,18,17,17,12,12,11,10,9,9,5,5,3,3,2,1);
var
  N, I, First, Last : Byte;
  A : integer;
  Found : boolean;
begin
  Writeln('Исходный массив:');
  for I := 1 to Count do Write(M[I]:3, ' '); Writeln; Writeln;
  Write('Введите значение элемента массива для поиска >');
  Readln(N);
  A:=0;
  First := 1;
  Last := Count;
  Found:=False;                        {Элемент не найден}
  repeat                               {Повторять поиск}
    I := (First + Last) div 2;          {Разделить на две части}
    if M[I] = N then Found:=True
    else
      begin
        if M[I] > N then First := I+1 {Искать элемент в правой части}
        else Last := I-1;             {Искать элемент в левой части}
      end;
    A:=A+1;                             {Увеличить счетчик числа итераций}
  until (Found) or (First>Last); {Завершить, если найдется искомый элемент или
  будет просмотрен весь массив}
  if Found then Writeln('Искомый элемент '.N, ' в массиве занимает '.I, '-ю
  позицию')
  else
    Writeln('В массиве нет искомого элемента '.N);
    Writeln('Поиск выполнен за '.A, ' итераций');
  end.
```

Запустите интегрированную среду программирования. Введите текст программы Find_Bin и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции запустите программу в пошаговом режиме и проследите за изменениями текущих значений переменных First, Last, I, M[I], а также значениями выражений $M[I] > N$ и ($Found$) or ($First>Last$). Обратите внимание на то, что бинарный поиск в сортированных массивах выполняется за небольшое число итераций.

Упражнение 5. Создадим программу, формирующую двумерный массив случайных чисел и вычисляющую значение среднего арифметического его элементов, больших, чем 20. Решение задачи сводится к последовательному перебору всех элементов массива с вычислением суммы тех элементов, значение которых больше 20, а по окончании их суммирования производится вычисление частного полученной суммы и количества элементов массива, удовлетворяющих условию суммирования.

В разделе описания программы опишем квадратный двумерный массив, указав в разделе описания констант количество строк **Strok = 15** и количество столбцов **Stolb = 15**, а в разделе описания переменных — массив целых чисел.

```
D : array[1..Strok,1..Stolb] of integer
```

Для указания положения элемента в массиве введем переменные:

I — номер строки, в которой расположен элемент;

J — номер столбца, в котором расположен элемент.

Для подсчета количества элементов массива, больших 20, введем целую переменную **K**, а для подсчета их суммы, а затем и значения среднего арифметического введем вещественную переменную **S**. Для заполнения массива случайными целыми числами применим функцию **Random** и поэтому в начале описания поместим подключение стандартного модуля **Crt**, в котором она содержится. Блок описания будет выглядеть так:

```
uses Crt;
const Strok=15;
      Stolb=15;
var
    D : array[1..Strok,1..Stolb] of integer;
    I, J, K : integer;
    S : real;
```

Так как массив **D** является двумерным, то мы реализуем присваивание случайных значений элементам массива в виде двух вложенных циклов **for**:

```
Randomize;
for I:= 1 to Strok do
  for J:= 1 to Stolb do D[I,J]:=Random(99);
```

Во внешнем цикле изменяется номер строки, во внутреннем — номер столбца. Данный фрагмент программы выполняется следующим образом: параметр **I** внешнего цикла принимает первоначальное значение 1, затем выполняется полный цикл изменения параметра **J** внутреннего цикла, и при этом элементам массива **D[1,J]** присваиваются значения результатов вычислений функции **Random(99)**. Аргумент 99 указывает интервал значений генерируемых случайных чисел от 0 до 99. После завершения внутреннего цикла параметр внешнего цикла **I** увеличивается на 1, проверяется условие **I <= Strok** и если оно выполняется, то выполняется тело внешнего цикла — внутренний цикл, т. е. выполняется полный цикл изменения параметра **J** внутреннего цикла при **I = 2**, и при этом элементам массива **D[2,J]** присваиваются значения результатов вычислений функции **Random(99)** и т. д., пока не завершится полный цикл изменений параметра внешнего цикла **I** от 1 до **Strok**. В результате всем элементам массива **D** будет присвоено значение результатов вычислений функции **Random(99)**.

Перед просмотром массива обнулим значения переменных **S** и **K**. Просмотр массива запишем в виде двух вложенных циклов, и так же, как и при инициализации массива, параметр внутреннего цикла будет определять номер столбца, в котором расположен элемент массива, а параметр внешнего цикла — номер строки. Поскольку тело и внешнего, и внутреннего цикла содержит более одного оператора, то для их записи используем составной оператор. В составном операторе тела внутреннего цикла выполняется печать элементов массива по столбцам в **I**-й строке и осуществляется последовательный поиск элементов массива, удовлетворяющих условию **D[I,J]>20**. Если элемент массива удовлетворяет данному условию, то значение суммы увеличивается на величину значения этого элемента, а счетчик числа таких элементов увеличивается на единицу. Для перехода к печати элементов массива, расположенных в следующей строке, используем оператор **Writeln** после тела внутреннего цикла. Данный фрагмент программы будет выглядеть следующим образом:

```
for I:= 1 to Strok do
begin
  for J:= 1 to Stolb do
  begin
    Write(D[I,J]:2,' ');
    if D[I,J]>20 then
    begin
      S:= S+ D[I,J]; {Суммирование подходящих элементов}
      K:=K+1;
    end;
  end;
  Writeln;
end;
```

После завершения просмотра всех элементов массива для вычисления значения среднего арифметического его элементов, больших 20, используем оператор присваивания **S:=S/K**.

В завершающей части программы поместим оператор вывода сообщения о результатах вычисления, использовав для этого стандартную процедуру **Writeln** с форматом **5 : 3** для вывода значения переменной **S**.

Полный текст программы решения задачи будет выглядеть таким образом:

```
program Sum_Mas_2;
uses Crt;
const Strok=15;
      Stolb=15;
var
    D : array[1..Strok,1..Stolb] of integer;
    I, J, K : integer;
    S : real;
begin
```

```

Randomize;           {Генерация массива случайных чисел}
for I:= 1 to Strok do
  for J:= 1 to Stolb do D[I,J]:=Random(99);
S:= 0;
K:= 0;
for I:= 1 to Strok do {Вычисление суммы элементов массива>20}
  begin
    for J:= 1 to Stolb do {Просмотр всех элементов в I-й строке}
      begin
        Write(D[I,J]:2,' ');
        if D[I,J]>20 then {Если найден элемент D[I,J]>20}
          begin
            S:= S+ D[I,J]; {Суммирование подходящих элементов}
            K:=K+1; {Увеличение счетчика подходящих элементов}
          end;
        end;
      end;
    WriteLn; {Переход к печати массива в следующей строке экрана}
  end;
  S:=S/K; {Расчет среднего арифметического}
  WriteLn('Среднее арифметическое элементов массива. больших 20 = ',S:5:3);
  ReadLn;
end.

```

Запустите интегрированную среду программирования. Введите текст программы Sum_Mas_2 и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. После успешного завершения компиляции проследите за работой программы в пошаговом режиме начиная с оператора S:=0 с просмотром текущих значений переменных I, J, K и S. Установите в строке if D[I,J]>20 then контрольную точку и запустите программу на выполнение до точки останова нажатием клавиши F4. После этого, нажимая F8, наблюдайте за изменениями значений переменных и выводом элементов массива на экран.

Контрольные вопросы и задания

Вопросы

1. Что такое массив?
2. Как определить местоположение элемента в массиве?
3. Что такое индекс? Каким требованиям он должен удовлетворять?
4. Опишите особенности расположения элементов массива в памяти компьютера. Каковы особенности размещения в памяти элементов многомерных массивов?
5. Каким образом задается описание массива, что в нем указывается?
6. Опишите общие и отличительные черты одномерных, двумерных и n-мерных массивов.

7. В каких операциях могут быть задействованы массивы и какие к ним при этом предъявляются требования?
8. Каким образом в Pascal задается обращение к элементу массива?
9. Почему при описании массивов предпочтительнее употреблять константы, а не указывать размеры массива в явном виде?
10. Что называют инициализацией массива, и зачем она применяется?
11. Что называется контрольной точкой? Как задать ее положение? Как осуществить выполнение программы до контрольной точки?
12. Что называется сортировкой массива? Какие методы сортировки вы знаете? Опишите их существенные отличия.
13. Как задать имена переменных или выражения для просмотра их значений в окне просмотра при выполнении программы по шагам?
14. Что понимается под поиском? Каковы отличительные черты линейного и бинарного поиска? В каких случаях целесообразнее использовать линейный поиск, а в каких — бинарный?

Задания

1. Введите с клавиатуры в массив X пять целочисленных значений, выведите их в одну строку через запятую; получите для массива среднее арифметическое.
2. Введите с клавиатуры пять целочисленных элементов массива X. Выведите на экран значения корней и квадратов для каждого из элементов массива.
3. Создайте массив из пяти фамилий и выведите их на экран в столбик, начиная с последней.
4. Создайте массив из пяти фамилий и выведите на экран те из них, которые начинаются с определенной буквы, которая вводится с клавиатуры.
5. Дан одномерный массив. Вставьте в него элемент L в позицию K.
6. Введите с клавиатуры целочисленные элементы матрицы 3×3, выведите исходную матрицу на экран. Умножьте каждый элемент матрицы на 3 и выведите результат на экран.
7. Создайте двумерный массив (20×15) целых чисел и найдите сумму всех его нечетных элементов.
8. Введите с клавиатуры целочисленные элементы матрицы 3×3 и вычислите сумму элементов каждого столбца.
9. Создайте матрицу 5×5, значение каждого элемента которой равно сумме номера строки и столбца, на пересечении которых он находится, и вычислите сумму элементов каждой строки.
10. Создайте массив из 15 целочисленных элементов и определите среди них минимальное значение.

11. Создайте двумерный массив X , имеющий четыре строки и три столбца и найдите в нем максимальный по абсолютному значению элемент, а также укажите номер строки и столбца, содержащих этот элемент. Например в массиве, показанном справа, максимальный по абсолютному значению элемент равен 8, он находится во второй строке третьего столбца.

2	1	3
-4	0	8
7	5	1
-3	1	0

12. Введите массив (не более 20 элементов) и определите, есть ли в нем элементы с одинаковыми значениями.
13. Напишите программу анализа значений температуры больного за сутки определите минимальное и максимальное значение, а также среднее арифметическое. Замеры температуры проводятся шесть раз, и результаты вводятся с клавиатуры в массив T .
14. Дана матрица A , имеющая N строк и N столбцов. Сформируйте два одномерных массива. В один запишите четные, а в другой — нечетные элементы матрицы. Выведите на экран все массивы.
15. Создайте двумерный массив вещественных чисел, имеющий 10 строк и 15 столбцов, выведите его на экран. Затем разделите каждый элемент массива на среднее арифметическое значение элементов строки, в которой они расположены, и результат выведите на экран.
16. Создайте матрицу из 15 строк и 15 столбцов. Вычислите произведение суммы элементов главной диагонали на сумму элементов i -й строки.
17. Сожмите одномерный массив, удалив элементы, предшествующие минимальному элементу.
18. Найдите в одномерном массиве два элемента, сумма которых максимальна, затем удалите все элементы, предшествующие тому элементу, у которого больше индекс.
19. Вычислите сумму элементов двумерного массива, индексы которых составляют в сумме заданное число K .
20. Создайте массив «шахматная доска».
21. Создайте одномерный массив, элементами которого являются суммы положительных элементов строк матрицы.
22. Найдите сумму элементов столбца и строки массива, на пересечении которых находится максимальный элемент.

23. Найдите сумму элементов массива, расположенных ниже главной диагонали, произведение не равных нулю элементов, расположенных выше главной диагонали и количество элементов в главной диагонали, попадающих в интервал $[-1; 1]$.
24. Найдите сумму минимальных элементов главной и побочной диагоналей.
25. Найдите длину наибольшего отрезка, соединяющего две точки с координатами, заданными таблицей $F(2, N)$.
26. Заданы координаты 40 точек на плоскости. Найдите расстояние до точки, наиболее удаленной от начала координат.
27. Задан список областных центров России. Присвойте переменной t название города с максимальным числом букв.
28. Сечение крыши имеет форму полукруга с радиусом R м. Требуется сформировать таблицу, содержащую длины опор, устанавливаемых через каждые $R/5$ м.
29. Таблица содержит 100 номеров выигрышных билетов. Проверьте, является ли билет с номером N выигрышным.
30. С 8 до 20 ч температура воздуха измерялась ежечасно. Известно, что в течение этого времени температура понижалась. Определите, в котором часу была впервые отмечена отрицательная температура.
31. В поликлинику поступили сведения о проживающих на обслуживаемой улице в виде таблиц, каждая из которых содержит номер дома, общее число жильцов дома, число детей и число пенсионеров. Всего поступили три такие таблицы. Требуется сформировать одну линейную таблицу, содержащую сведения о трех домах.
32. Дан список учеников класса и оценки каждого из учеников за выполнение двух контрольных работ. Требуется:
- 1) подсчитать число учеников, выполнивших первую работу на 5;
 - 2) подсчитать число учеников, выполнивших хотя бы одну работу на 5;
 - 3) подсчитать число учеников, выполнивших обе работы на 5;
 - 4) подсчитать число учеников, выполнивших вторую работу на 4 и 5;
 - 5) подсчитать число учеников, выполнивших обе работы на 4 и 5;
 - 6) найти число учеников, выполнивших обе работы на 5, число учеников, выполнивших обе работы на 4, и число учеников, не выполнивших обе работы;
 - 7) найти число учеников, написавших хотя бы одну из двух работ на 5, и число учеников, не написавших хотя бы одну работу;
 - 8) вывести список учеников, выполнивших первую работу на 5;
 - 9) вывести список учеников, не выполнивших ни одной работы;
 - 10) вывести список учеников, не выполнивших хотя бы одну работу.

33. В расписании движения поездов по станции Масловка указаны: номера поездов, пункты следования, время прибытия и отправления, направления следования (южное, северное, западное, восточное). Сколько поездов следует в каждом из направлений?
34. Дан список футбольных команд Российской Премьер-лиги и количество очков, набранных каждой командой в чемпионате России. Известно, что нет команд с равным числом очков. Какая из команд стала чемпионом? Составьте список команд, набравших более 15 очков.
35. Дана таблица названий товаров, выпускаемых заводом. Определите, повторяется ли в этой таблице название первого товара, и, если повторяется, удалите название первого товара из таблицы.
36. Дан список фамилий брокеров товарной биржи из N человек. Поменяйте местами фамилии брокеров: первого и последнего, второго и предпоследнего, третьего от начала и третьего от конца и т. д.
37. Даны две таблицы. Одна содержит наименования услуг, выполняемых в доме быта, а другая — расценки за эти услуги. Удалите из обеих таблиц все, что предшествует услуге, цена которой составляет P руб.
38. Объедините две линейные таблицы A и B в новую таблицу C , поставив элементы таблицы A на нечетные места, а элементы таблицы — B на четные.
39. В линейном массиве найдите максимальный элемент. Вставьте порядковый номер максимального элемента за ним, передвинув все оставшиеся элементы на одну позицию вправо.
40. В линейном массиве найдите индексы тех двух элементов, сумма которых максимальна среди сумм всевозможных пар. Удалите все элементы массива, предшествующие тому элементу, индекс которого является наибольшим из двух найденных. Преобразованный массив выведите на экран.
41. В квадратной таблице поменяйте местами элементы строки и столбца, на пересечении которых находится минимальный из положительных элементов.
42. Наименьший элемент каждой строки прямоугольной таблицы, начиная со второй, замените наибольшим элементом предшествующей строки.
43. Даны две таблицы из N слов различной длины. Упорядочите слова по возрастанию их длин.
44. Даны стоимость различных деталей, выпускаемых мастерской, и их названия. Отсортируйте их по стоимости и по алфавиту.
45. Задана таблица из N чисел. Сколько треугольников можно составить из этих чисел? Найдите треугольник с максимальной площадью.

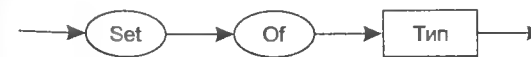
Глава 8

Множества и записи

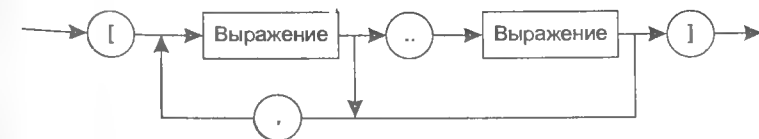
Описание типа «множество»

Множество — это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое. Каждый объект в множестве называется *элементом множества*. Все элементы множества должны принадлежать одному из скалярных типов, кроме вещественного. Этот тип называется *базовым типом* множества. Базовый тип задается диапазоном или перечислением. Область значений типа «множество» — набор всевозможных подмножеств, составленных из элементов базового типа. В выражениях на языке Pascal значения элементов множества указываются в квадратных скобках: $[1,2,3,4]$, $['a','b','c']$, $['a'..'z']$. Если множество не имеет элементов, оно называется *пустым* и обозначается как $[\]$. Количество элементов множества называется его *мощностью*.

Для описания множественного типа используется словосочетание *set of* (множество из...). Синтаксическая диаграмма множественных типов имеет следующий вид:



Изображение множества:



Исходя из синтаксической диаграммы, представим формат записи множественных типов:

type

<имя типа> = set of <элемент1.....элементn>;

var

<идентификатор.....> : <имя типа>;

Можно задать множественный тип и без предварительного описания:

```
var
  <идентификатор....> : set of <элемент1....>;
```

Пример:

```
type
  Simply = set of 'a'..'h';
  Number = set of 1..31;
var
  Pr : Simply;
  N : Number;
  Letter : set of char; {Определение множества без предварительного описания
в разделе типов}
```

В данном примере переменная Pr может принимать в качестве значений символы латинского алфавита от 'a' до 'h'; N — любое значение в диапазоне 1..31; Letter — любой символ. Попытка присвоить другие значения вызовет программное прерывание. Количество элементов множества не должно превышать 256, соответственно номера значений базового типа должны находиться в диапазоне 0..255. Контроль диапазонов осуществляется включением директивы {\$R+}. Объем памяти, занимаемый одним элементом множества, составляет 1 бит. Объем памяти для переменной типа «множество» вычисляется по формуле:

Объем памяти = (Max DIV 8) – (Min DIV 8) + 1,

где Max и Min — верхняя и нижняя границы базового типа.

Операции над множествами

При работе с множествами допускается использование операций отношения =, <>, >=, <=, объединения, пересечения, разности множеств и операции in. Результатом выражений с применением этих операций является значение True или False.

Операция «равно» (=). Два множества A и B считаются равными, если они состоят из одних и тех же элементов. Порядок следования элементов в сравниваемых множествах значения не имеет.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3,4]	[1,2,3,4]	A = B	True
['a','b','c']	['c','a']	A = B	False
['a'..'z']	['z'..'a']	A = B	True

Операция «не равно» (<>). Два множества A и B считаются неравными, если они отличаются по мощности или по значению хотя бы одного элемента.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3]	[3,1,2,4]	A <> B	True
['a'..'z']	['b'..'z']	A <> B	True
['c'..'v']	['t'..'c']	A <> B	False

Операция «больше или равно» (>=). Эта операция используется для определения принадлежности множеств. Результат операции A >= B равен True, если все элементы множества B содержатся в множестве A. В противном случае результат равен False.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3,4]	[2,3,4]	A >= B	True
['a'..'z']	['b'..'t']	A >= B	True
['z','x','c']	['c','x']	A >= B	True

Операция «меньше или равно» (<=). Эта операция используется аналогично предыдущей операции, но результат выражения A <= B равен True, если все элементы множества A содержатся в множестве B. В противном случае результат равен False.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3]	[1,2,3,4]	A <= B	True
['d'..'h']	['z'..'a']	A <= B	True
['a','v']	['a','n','v']	A <= B	True

Операция in. Эта операция используется для проверки принадлежности какого-либо значения указанному множеству. Обычно применяется в условных операторах.

Например:

Значение A	Выражение	Результат
2	if A in [1,2,3] then...	True
v	if A in ['a'..'n'] then...	False
x1	if A in [x0, x1, x2, x3] then...	True

При использовании операции in проверяемое на принадлежность значение и множество в квадратных скобках не обязательно предварительно описывать в разделе описаний. Операция in позволяет эффективно и наглядно производить сложные проверки условий, заменяя иногда десятки других операций.

Например, выражение `if (a=1) or (a=2) or (a=3) or (a=4) or (a=5) or (a=6) then` можно заменить более коротким выражением `if a in [1..6] then...`

Часто операцию `in` пытаются записать с отрицанием: `X NOT in M`. Такая запись является ошибочной, так как две операции следуют подряд; правильная конструкция имеет вид `NOT (X in M)`.

Объединение множеств (+). Объединением двух множеств является третье множество, содержащее элементы обоих множеств.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3]	[1,4,5]	$A + B$	[1,2,3,4,5]
['A'..'D']	['E'..'Z']	$A + B$	['A'..'Z']
[]	[]	$A + B$	[]

Пересечение множеств (*). Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3]	[1,4,2,5]	$A * B$	[1,2]
['A'..'Z']	['B'..'R']	$A * B$	['B'..'R']
[]	[]	$A * B$	[]

Разность множеств (-). Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество.

Например:

Значение A	Значение B	Выражение	Результат
[1,2,3,4]	[3,4,1]	$A - B$	[2]
['A'..'Z']	['D'..'Z']	$A - B$	['A'..'C']
[X1,X2,X3,X4]	[X4,X1]	$A - B$	[X2,X3]

Результат операций над двумя множествами можно наглядно представить помощью закрашенных частей двух прямоугольников:



Использование в программе данных типа `set` дает ряд преимуществ: значительно упрощаются сложные операторы `if`, увеличивается степень наглядности программы и понимания алгоритма решения задачи, экономятся память, время компиляции и выполнения. Имеются и отрицательные моменты, основной из них — отсутствие в языке Pascal средств ввода-вывода элементов множества, поэтому программист сам должен писать соответствующие процедуры.

Упражнение 1. Иллюстрацией описания множеств и операций над ними может служить программа `Dem_Mno`, в которой описаны множества чисел `D`, `D1`, `D2`, `D3`. Затем они заполнены следующим образом: множество `D1` — четными числами 2, 4, 6, 8; множество `D2` — числами 0, 1, 2, 3, 5; множество `D3` — нечетными числами 1, 3, 5, 7, 9. После этого над множествами выполнены операции объединения, разности и пересечения.

```

program Dem_Mno; {Демонстрация операций над множествами}
type
  Digits = set of 0..9;
var
  D1, D2, D3, D : Digits;
begin
  D1 := [2,4,6,8];      {Заполнение множеств}
  D2 := [0..3,5];
  D3 := [1,3,5,7,9];
  D := D1 + D2;         {Объединение множеств D1 и D2}
  D := D + D3;          {Объединение множеств D и D3}
  D := D - D2;          {Разность множеств D и D2}
  D := D * D1;          {Пересечение множеств D и D1}
end.

```

Как видно из текста программы, сначала описан тип `Digits = set of 0..9`, затем описаны переменные `D1`, `D2`, `D3`, `D` этого типа. В первой части программы осуществляется заполнение множеств, а затем над множествами выполняются операции объединения, пересечения, разности.

Для проверки работы программы запустите интегрированную среду программирования. Введите текст программы `Dem_Mno` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Так как в Pascal отсутствуют средства ввода-вывода элементов множества, то работу программы придется проверять, выполняя ее в пошаговом режиме и отслеживая изменения значений переменных `D1`, `D2`, `D3`, `D` в окне просмотра.

Упражнение 2. Опишите множество `M` (1..50). Сделайте его пустым. Вводя целые числа с клавиатуры, заполните множество 10 элементами. В разделе описания переменных опишем множество целых чисел от 1 до 50, переменную `X` целого типа будем использовать для считывания числа-кан-

дидата в множество, целую переменную I используем для подсчета количества введенных чисел.

В начале программы применим операцию инициализации множества $M := []$, так как оно не имеет элементов и является пустым.

Заполнение множества элементами произведем с использованием оператора for, параметр которого I будет указывать порядковый номер вводимого элемента. Операцию заполнения множества запишем в виде оператора присваивания $M := M + [X]$. Контроль заполнения множества запишем с использованием операции проверки принадлежности in. Если условие $X \text{ in } M$ выполняется, выведем сообщение о том, что число X помещено в множество.

Текст программы описания и заполнения множества будет таким:

```
program Inpu_Mno;
var
  M : set of 1..50;
  X, I : integer;
begin
  M := [];           {M — пустое множество}
  for I := 1 to 10 do
  begin
    Write('Введите ', I, ' -й элемент множества: ');
    Readln(X);
    if (X in M) then {Если введенное число входит в множество M}
    begin
      Writeln(X, ' помещен в множество 1..50');
      M := M + [X];
    end;
  end;
  Writeln;
end.
```

Для проверки работы программы запустите интегрированную среду программирования. Введите текст программы Inpu_Mno и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Так как в Pascal отсутствуют средства ввода-вывода элементов множества, то работу программы придется проверять, выполняя ее в пошаговом режиме и отслеживая изменения значений переменных I, X и M в окне просмотра. Попробуйте задать значения числа X, большие 50, повторно задавать одинаковые значения X и проанализировать значения множества M.

Упражнение 3. Описать множества гласных и согласных букв русского алфавита, определить количество гласных и согласных букв в предложении, введенном с клавиатуры.

Зададим тип Letters — множество букв русского алфавита, затем опишем переменные этого типа: Glasn — множество гласных букв, Sogl — множество

согласных букв. Вводимое с клавиатуры предложение опишем переменной Text типа string. Для указания символа в строке Text применим переменную I типа byte. Для подсчета количества гласных и согласных букв опишем переменные G и S. Блок описания программы будет выглядеть следующим образом:

```
type
  Letters = set of 'A' .. 'Я';
var
  Glasn, Sogl : Letters;
  Text : string;
  I : byte;
  G, S : integer;
```

В начале программы поместим заполнение множеств гласных и согласных букв:

```
Glasn := ['А', 'а', 'Е', 'е', 'И', 'и', 'О', 'о', 'У', 'у', 'Э', 'э', 'Ю', 'ю', 'Я', 'я'];
Sogl := ['Б', 'б', 'Д', 'д', 'Ж', 'ж', 'З', 'з', 'К', 'к', 'Н', 'н', 'П', 'п', 'Т', 'т', 'Ф', 'ф', 'Ц', 'ц', 'Ч', 'ч', 'Ш', 'ш', 'Щ', 'щ', 'Ъ', 'ъ', 'Ь', 'ь', 'Ы', 'ы', 'Э', 'э', 'Ю', 'ю', 'Я', 'я'];
```

После этого реализуем вывод приглашения на ввод предложения и считывание этого предложения в переменную Text.

```
Write('Введите предложение ');
Readln(Text);
```

Перед началом подсчета количества гласных и согласных букв в предложении обнулим значения переменных G и S.

Проверку принадлежности символов, составляющих предложение, множествам гласных или согласных букв русского алфавита реализуем с использованием оператора for, параметр I которого, изменяясь от 1 до значения длины предложения, будет указывать порядковый номер символа в предложении. Принадлежность очередного символа предложения множеству гласных или согласных букв запишем в виде операции in. Если условие $\text{Text}[I] \text{ in } \text{Glasn}$ выполняется, то счетчик гласных букв G увеличивается на 1. Если выполняется условие $\text{Text}[I] \text{ in } \text{Sogl}$, тогда увеличивается на 1 счетчик согласных букв S. Если не выполняется ни первое, ни второе условие, значит, очередной символ в предложении не является буквой русского алфавита. Данный фрагмент программы будет выглядеть так:

```
for I := 1 to Length(Text) do
begin
  if Text[I] in Glasn then G := G + 1;
  if Text[I] in Sogl then S := S + 1;
end;
```

В заключительной части программы поместим вывод сообщения о результатах подсчета гласных и согласных букв в предложении.

В целом текст программы будет выглядеть следующим образом:

```
program Glasn_Sogl; {Подсчет гласных и согласных букв в предложении}
type
  Letters = set of 'A'..'я';
var
  Glasn, Sogl : Letters;
  Text : string;
  I : byte;
  G, S : byte;
begin
  Glasn:=['A','a','E','e','И','и','O','o','У','у','Э','э','Ю','ю','Я','я',
  Sogl:=['Б','Д','б','д','Ж','ж','З','з','К','к','Н','н','П','п','Т','т',
  'Щ','щ','Ф','ф','Х','х','Ц','ц'];
  Write('Введите предложение ');
  ReadLn(Text);
  G:=0;
  S:=0;
  for I:=1 to Length(Text) do
    begin
      if Text[I] in Glasn then G:=G+1;
      if Text[I] in Sogl then S:=S+1;
    end;
  WriteLn('В предложении "',Text,'" 'G,' гласных и 'S,' согласных букв');
end.
```

Запустите интегрированную среду программирования. Введите текст программы Glasn_Sogl и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Проверьте работу программы, вводя различные предложения.

Упражнение 4. Создать программу с контролем ввода данных, обеспечив ввод фамилии, имени и отчества только на русском языке.

В разделе описания переменных опишем множество Litera, включающее в себя все допустимые символы, переменную Name для хранения Ф.И.О., Ch для посимвольного ввода Ф.И.О. Переменная логического типа Rus будет указывать, принадлежит ли очередной вводимый символ множеству букв русского алфавита. Для использования стандартной функции чтения кода нажатой клавиши ReadKey подключим модуль Crt. Блок описания будет выглядеть так:

```
uses Crt;
var
  Litera : set of char;
  Name : string;
  Ch : char;
  Rus : boolean;
```

В начале программы произведем заполнение множества букв русского алфавита:

```
Litera:=[' ','A'..'Я','a'..'я'];
```

Затем запишем вывод запроса о вводе Ф.И.О.:

```
Write('Введите фамилию, имя, отчество ');
```

Считывание всех символов, входящих в запись Ф.И.О., запишем с применением оператора repeat. Завершение цикла будем производить по нажатии клавиши Enter (код 13).

Считывание каждого отдельного символа вводимого текста Ф.И.О. запишем в виде оператора repeat, условием завершения которого будет значение True переменной Rus. В теле оператора повтора сначала считаем в переменную Ch код нажатой клавиши, затем проверим, входит ли введенный символ в множество букв русского алфавита. Если условие Ch in Litera выполняется, то переменной Rus присваивается значение True, символ добавляется к строке Name и печатается в строке ввода, после чего управление передается на оператор until. Так как условие завершения цикла ввода буквы русского алфавита выполняется, осуществляется переход к считыванию кода нажатой клавиши.

Если очередной нажатой клавишей является Enter, то ReadKey возвращает код 13. Если в операторе if Ch=#13 условие не выполняется, то управление передается за его пределы, т. е. на оператор until Ch=#13, и ввод строки Name завершается, так как нажата клавиша Enter.

Данный блок программы будет выглядеть так:

```
repeat {Считать всю строку Name}
  repeat {Считать один символ, входящий в множество Litera}
    Ch:=ReadKey; {Считать в Ch код нажатой клавиши}
    if Ch=#13 then
      begin
        Rus:=Ch in Litera;
        if Rus {Если символ входит в множество Litera}
          then
            begin
              Name:=Name+Ch; {Приклеить введенный символ к Name}
              Write(Ch); {Напечатать введенный символ в строке ввода}
            end
          else {Код нажатой клавиши не входит в множество Litera}
            begin
              WriteLn('Переключитесь в русский регистр');
              Write('и введите Ваше имя ');
            end;
          end;
        until Rus; {Завершить ввод очередного символа на русском языке}
      until Ch=#13; {Завершить ввод строки Name т.к. нажата клавиша Enter}
```

В заключительной части программы выполняется перевод курсора на следующую строку и вывод текста «Здравствуйте», после чего печатается значение переменной Name.

Полный текст программы будет выглядеть следующим образом:

```
program FOI_Rus;      {Ввод Ф.И.О. только на русском языке}
uses Crt;
var
  Litera : set of char;
  Name : string;
  Ch : char;
  Rus : boolean;
begin
  Litera:=[' ','A'..'n','p'..'я'];
  Write('Введите фамилию, имя, отчество ');
  repeat
    {Считать всю строку Name}
    repeat
      {Считать один символ, входящий в множество Litera}
      Ch:=ReadKey; {Считать в Ch код нажатой клавиши}
      if Ch<>#13 then begin
        Rus:=Ch in Litera;
        if Rus {Если символ входит в множество Litera}
        then
          begin
            Name:=Name+Ch; {Приклеить введенный символ к Name}
            Write(Ch); {Напечатать введенный символ в строке ввода}
          end
        else {Код нажатой клавиши не входит в множество Litera}
          begin
            WriteLn('Переключитесь в русский регистр');
            Write('и введите Ваше имя ');
          end;
        end;
      until Rus; {Завершить ввод очередного символа на русском языке}
    until Ch=#13; {Завершить ввод строки Name т.к. нажата клавиша Enter}
    WriteLn;
    WriteLn('Здравствуйте, ',Name);
  end.
```

Запустите интегрированную среду программирования. Введите текст программы FOI_Rus и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Проверьте работу программы, вводя латинские и русские символы. Проследите в пошаговом режиме за значениями переменных Ch, Name, Rus и выражений Ch=#13, Ch in Litera.

Контрольные вопросы и задания

Вопросы

1. Что такое множество? Каким требованиям должны удовлетворять элементы множества? В чем состоят преимущества использования типа «множество»?
2. Что такое базовый тип множества? Как он задается?

3. Какое множество называется пустым, как оно обозначается?
4. Как задается описание множественного типа?
5. Какие операции допустимы над множествами? Каков тип результатов выражений с применением операций над множествами?
6. Какие множества считаются равными, неравными? Имеет ли значение для сравниваемых множеств порядок следования элементов?
7. Для чего применяются операции «больше или равно», «меньше или равно»? В чем их отличие?
8. Для чего применяется операция in? Опишите особенности ее применения.
9. Что называется объединением множеств?
10. Что называется пересечением множеств?
11. Что называется разностью множеств?

Задания

1. Опишите множества M1 (1..10) и M2 (20..30).
2. Опишите множества R и L, содержащие русские и латинские буквы.
3. Опишите множество Pr (1..20) и поместите в него все простые числа в диапазоне 1..20.
4. Опишите множество Alf ('a'..'я') и поместите в него гласные буквы.
5. Опишите множества M1 (1,2) и M2 (2,1). Сравните множества M1 и M2 на равенство.
6. Опишите множества M1 ('a','b') и M2 ('b','a','c'). Сравните два этих множества на равенство.
7. Опишите множества M1 ('a','b','c') и M2 ('a','c'). Сравните два этих множества с использованием операции >=.
8. Опишите множества M1 (1,2,3) и M2 (1,2,3,4). Сравните два этих множества с использованием операции <=.
9. Опишите множества M1 (1,2) и M2 (5,6). Получите результирующее множество M3 = M1+M2. Определите, содержится ли в M3 элемент 7.
10. Опишите множества M1 (1,2,3,4) и M2 (3,4,1). Получите результирующее множество M3 = M1-M2. Определите, содержится ли в M3 элемент 2.
11. Опишите множества M1 (1,2,3) и M2 (1,4,2,5). Получите результирующее множество M3 = M1*M2. Определите, содержатся ли в M3 элементы 1 и 2.
12. Опишите множества R и L, содержащие русские и латинские буквы. В цикле вводите русские и латинские буквы и выводите соответствующее сообщение. Выход из цикла — введенная буква Z.
13. Опишите множество Pr (1..20) и поместите в него все простые числа в диапазоне 1..20. В цикле организуйте ввод чисел в диапазоне 1..20 и определите, простые они или нет. Выход из цикла — введенное значение, равное 99.

14. Имеется множество **Lat** ('a'..'z'). Придумайте простейший способ для вывода на печать его содержимого.
15. Напишите программу вычисления суммы мест, на которых в слове **X** стоят гласные буквы.

Записи

В информационно-поисковых системах (таких, как адресное бюро, телефонная справочная служба и т. п.) приходится хранить и обрабатывать большие объемы данных. Как было показано в гл. 7, при решении научно-технических и экономических задач обработки совокупностей большого количества значений используются массивы. Но при работе с массивами основное ограничение заключается в том, что все элементы массива должны иметь один и тот же тип данных. Именно поэтому, решая задачу формирования двумерного массива данных о фамилиях учащихся и их возрасте, мы объявляли и фамилии, и возраст учащихся величинами типа **string**, а для преобразования значения возраста учащегося из типа **string** в величину целочисленного типа использовали стандартную функцию **Val**. Решение задачи получалось громоздким, затрачивалось время на преобразование типа.

Иногда для решения задач, в которых возникает необходимость хранить и обрабатывать совокупности данных различного типа, используются отдельные массивы для каждого типа данных, а для установления соответствия между ними вводятся соответствующие индексы.

Итак, реальные данные об объектах часто описываются величинами разных типов. Например, товар на складе описывается следующими величинами: наименование, количество, цена, наличие сертификата качества и т. д. В этом примере наименование — величина типа **string**, количество — **integer**, цена — **real**, наличие сертификата качества можно описать величиной типа **boolean**. Для записи комбинации объектов разных типов в **Pascal** применяется комбинированный тип данных — запись.

Запись представляет собой наиболее общий и гибкий структурированный тип данных, так как она может быть образована из неоднотипных компонентов и в ней явным образом выражена связь между элементами данных, характеризующими реальный объект.

Описание типа «запись»

Запись — это структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов. Определение типа записи начинается идентификатором **record** и заканчивается зарезервированным словом **end**. Между ними располагается список компонентов, называемых полями, с указанием идентификаторов полей и типа каждого поля.

Формат:

```
type
  <имя типа> = record
    <идентификатор поля>: <тип компонента>;

    <идентификатор поля>: <тип компонента>

  end;
var
  <идентификатор...> : <имя типа>;
```

Пример:

```
type
  Car = record
    Number : integer;      {Номер}
    Marka : string[20];    {Марка автомобиля}
    FIO : string[40];       {Фамилия, инициалы владельца}
    Address : string[60]    {Адрес владельца}
  end;
var
  M, V : Car;
```

В данном примере запись **Car** содержит четыре компонента: номер, название марки машины, фамилию владельца и его адрес. Доступ к полям записи осуществляется через переменную типа «запись». В нашем случае это переменные **M** и **V** типа **Car**.

Идентификатор поля должен быть уникален только в пределах записи, однако во избежание ошибок лучше делать его уникальным в пределах всей программы. Объем памяти, необходимый для записи, складывается из длин полей.

Значения полей записи могут использоваться в выражениях. Имена отдельных полей не применяются по аналогии с идентификаторами переменных, поскольку может быть несколько записей одинакового типа. Обращение к значению поля осуществляется с помощью идентификатора переменной и идентификатора поля, разделенных точкой. Такая комбинация называется *составным именем*. Например, чтобы получить доступ к полям записи **Car**, надо записать:

```
M.Number, M.Marka, M.FIO, M.Address
```

Составное имя можно использовать везде, где допустимо применение типа поля. Для присваивания полям значений используется оператор присваивания.

Пример:

```
M.Number := 1678;
M.Marka := 'ГАЗ - 24';
```



```
M.FIO := 'Демьяшкин В.А.';
M.Address := 'ул. Пушкина 12 - 31';
```

Составные имена можно использовать, в частности, в операторах ввода-вывода.

```
Read(M.Number, M.Marka, M.FIO, M.Address);
Write(M.Number:4, M.Marka:7, M.FIO:12, M.Address:25);
```

Допускается применение оператора присваивания и к записям в целом, если они имеют одинаковый тип. Например,

```
V := M;
```

После выполнения этого оператора значения полей записи V станут равны значениям соответствующих полей записи M.

В ряде задач удобно пользоваться массивами из записей. Их можно описать следующим образом:

```
type
  Person = record
    FIO : string[20];
    Age : 1 .. 99;
    Prof : string[30]
  end;
var
  List : array[1..50] of Person;
```

Обращение к полям записи имеет несколько громоздкий вид, что особенно неудобно при использовании мнемонических идентификаторов длиной более пяти символов. Для решения этой проблемы в языке Pascal предназначен оператор with, который имеет следующий формат:

```
with <переменная типа запись> do <оператор>;
```

Один раз указав переменную типа запись в операторе with, можно работать с именами полей как с обычными переменными, т. е. без указания перед идентификатором поля имени переменной, определяющей запись.

Пример:

Присвоить значения полям записи Car с помощью оператора with.

```
with M do
begin
  Number := 2347;
  Marka := 'ГАЗ - 24';
  FIO := 'Петров В.И.';
  Address := 'ул. Остужева, 5';
end;
```

Pascal допускает вложение записей (т. е. поле записи может, в свою очередь, тоже являться записью), соответственно оператор with тоже может быть вложенным:

```
with RV1 do
  with RV2 do
    with RVn do
      ...
    end;
  end;
end;
```

что эквивалентно конструкции

```
with RV1, RV2, ... RVn do
  ...
end;
```

ВНИМАНИЕ

Уровень вложенности записей не должен превышать 9.

Обычно записи используются при работе с динамическими структурами и для организации файлов на дисках. Записи могут служить также для описания комплексных чисел, так как в Pascal нет для этого специальных средств. В этом случае действительная и мнимая части комплексного числа являются полями записи, например:

```
type
  Complex = record
    RealPart: real; {Действительная часть}
    ImagePart: real; {Мнимая часть}
  end;
var
  A, B, C : Complex; {A, B, C - переменные типа Complex}
begin
  A.RealPart := 6.3;
  A.ImagePart := 1.9;
  ...
```

Во второй части книги вы познакомитесь с объектами, описание свойств которых также можно представить в виде записи.

Записи с вариантами

Записи, представленные выше, имеют строго определенную структуру. В некоторых случаях это серьезно ограничивает возможности их применения. Поэтому в Pascal имеется возможность задать тип записи, содержащий произвольное число вариантов структуры. Такие записи называются *записями с вариантами*. Записи с вариантами обеспечивают средства объединения записей, которые похожи, но не идентичны по форме. Они состоят из фиксированной и вариантной частей.

Использование фиксированной части не отличается от описанного ранее. Вариантная часть формируется с помощью оператора case. Он задает особое поле записи — поле признака, которое определяет, какой из вариантов в данный момент будет активизирован. Значением признака в каждый текущий момент выполнения программы должна быть одна из расположенных следом

констант. Константа, служащая признаком, задает вариант записи и называется *константой выбора*. Формат:

```
type
  Rec = record
    case <поле признака> : <имя типа> of
      <константа выбора1> : (поле...:тип);
      ...
      <константа выбораn> : (поле...:тип)
    end;
```

Компоненты каждого варианта (идентификаторы полей и их типы) заключаются в круглые скобки. У части *case* нет отдельного *end*, как этого следовало бы ожидать по аналогии с оператором *case*. Одно слово *end* заканчивает всю конструкцию записи с вариантами. Необходимо отметить, что количество полей каждого из вариантов не ограничено.

Объем памяти, необходимый для записи с вариантами, складывается из длины полей фиксированной части и максимального по длине поля переменной части.

Пример:

```
type
  Rec = record
    Number . byte;
    Code . integer;
    case Flag . boolean of
      True . (Price1 : integer);
      False . (Price2 : real)
    end;
  var
    PRec : Rec;
```

Поля *Number* и *Code* расположены в фиксированной части записи, они доступны в программе в любой момент независимо от значения поля признака *Flag*. Поле *Price1* может использоваться только в том случае, если значение поля признака *Flag* равно *True*. Поле *Price2* доступно в противоположном случае, т. е. если значение *Flag* равно *False*.

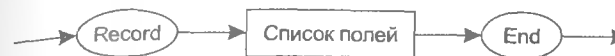
СОВЕТ

При использовании записей с вариантами необходимо придерживаться следующих правил:

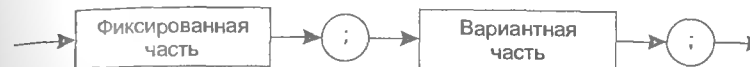
- все имена полей должны отличаться друг от друга по крайней мере одним символом, даже если они встречаются в разных вариантах;
- запись может иметь только одну вариантную часть, причем она должна размещаться в конце записи;
- если поле, соответствующее какой-либо метке, является пустым, то оно записывается следующим образом: <метка>: ();

Обобщая, можно записать синтаксическую диаграмму определения типа записи следующим образом:

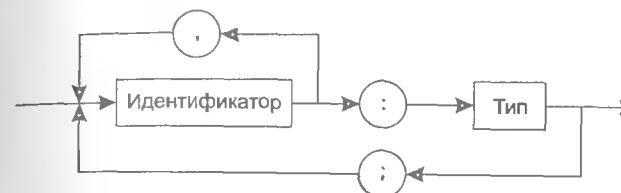
Запись:



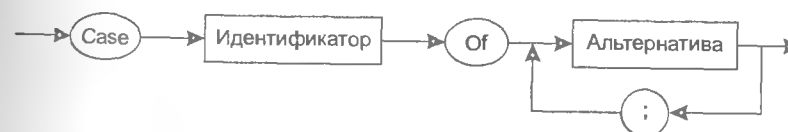
Список полей:



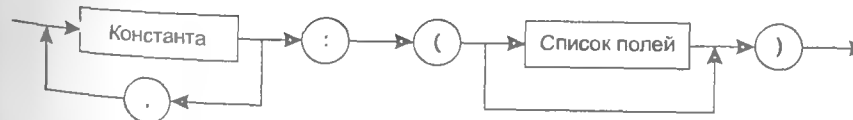
Фиксированная часть:



Вариантная часть:



Альтернатива:



Упражнение 5. Создадим программу, организующую ввод следующих данных об учащихся: имя, фамилия, возраст, школа, класс, и записывающую их в массив записей, а затем выводящую сведения об учащихся по номеру записи и по номеру класса.

Так как сведения об учащихся представляют собой данные разных типов, то для их описания в разделе типов введем тип *Record_Type* — запись следующей структуры:

```
Record_Type = Record
  Name . String[10];
  SurName . String[20];
  Age . Byte;
```

```
School : Integer;
Class  : Byte
```

end;

где Name — поле для записи имени учащегося (строка до 10 символов);

SurName — поле для записи фамилии учащегося (строка до 20 символов);

Age — поле для записи возраста (так как значение возраста не превысит 255 тип поля можно описать как byte);

School — поле для записи номера школы (целое число);

Class — поле для записи номера класса (целое число).

В разделе описания переменных опишем массив Record_Array, состоящий из 10 записей об учащихся описанного выше типа Record_Type:

```
Record_Array : Array[1 .. 10] of Record_Type.
```

Для указания номера элемента массива введем переменную Number_of_Array, принимающую целые значения от 1 до 10. Для поиска учащихся введем целочисленную переменную Class, которая будет принимать значения номера класса, заданного с клавиатуры.

Для ввода данных об учащихся напомним процедуру, в которой выводятся запросы на ввод данных и считываются значения этих данных в соответствующие поля записи. Для обращения к полям записи используем точечную нотацию, т. е. будем указывать имя записи, а затем, после точки, — имя поля например:

```
Record_Array[Number_of_Array].Name.
```

Процедура ввода данных в элемент с номером Number_of_Array массива записей Record_Array будет выглядеть следующим образом:

```
procedure Input_Data;
```

```
begin
```

```
  WriteLn ('Введите данные N°', Number_of_Array, ' :');
```

```
  Write (' Ваше имя ? ');
```

```
  ReadLn (Record_Array[Number_of_Array].Name);
```

```
  Write (' Фамилия ? ');
```

```
  ReadLn (Record_Array[Number_of_Array].SurName);
```

```
  Write (' Ваш возраст ? ');
```

```
  ReadLn (Record_Array[Number_of_Array].Age);
```

```
  Write (' Школа ? ');
```

```
  ReadLn (Record_Array[Number_of_Array].Class);
```

```
  Write (' и класс ? ');
```

```
  ReadLn (Record_Array[Number_of_Array].School);
```

```
  WriteLn
```

```
end;
```

Вывод на экран информации, хранящейся в одном элементе массива записей, реализуем с использованием предложения with ... do, в котором укажем имя текущей записи — элемент с номером Number_of_Array массива записей Record_Array[Number_of_Array], вследствие чего оно будет автоматически присоединено ко всем именам полей, упоминаемым в теле предложения with.

Эта процедура будет выглядеть следующим образом:

```
procedure Write_Data;
```

```
begin{Начало процедуры вывода на экран содержимого текущей записи}
```

```
  with Record_Array[Number_of_Array] do
```

```
    begin{Тело предложения with Record_Array [Number_of_Array] do}
```

```
      WriteLn (' Имя : ', Name);
```

```
      WriteLn (' Фамилия : ', SurName);
```

```
      WriteLn (' Возраст : ', Age);
```

```
      WriteLn (' Школа : ', School);
```

```
      WriteLn (' Класс : ', Class);
```

```
    end; {Конец тела предложения with}
```

```
end; {Конец процедуры}
```

В первой части основной программы применим оператор for для записи данных о десяти учащихся в массив записей. В теле этого оператора вызывается процедура Input_Data, которая вводит данные в элемент массива с номером Number_of_Array.

```
for Number_of_Array:=1 to 10 do
```

```
  Input_Data;
```

```
WriteLn;
```

Для вывода сведений об учащемся по номеру записи присвоим значение 5 номеру записи Number_of_Array в массиве данных и запишем вызов процедуры Write_Data. Этот фрагмент программы будет выглядеть так:

```
WriteLn ('Вывожу данные о пятом ученике : ');
```

```
WriteLn;
```

```
Number_of_Array:=5;
```

```
Write_Data;
```

Для поиска учащегося по номеру класса запишем вывод запроса на ввод искомого номера класса и считывание его значения в переменную Class. После этого при помощи оператора for, изменяющего значение параметра Number_of_Array от 1 до 10, выполним просмотр всех элементов массива записей, сравнивая значение переменной Class со значением соответствующего поля текущей записи.

Если условие Record_Array[Number_of_Array].Class=Class выполняется, значит, найдена запись об учащемся, удовлетворяющая условию поиска, поэтому после слова then разместим вызов процедуры Write_Data, которая будет выводить значения полей данной записи на экран.

Соответствующий фрагмент программы будет выглядеть следующим образом:

```
WriteLn ('Вывожу данные по номеру класса : ');
WriteLn;
WriteLn (' Введите номер класса : ');
ReadLn (Class);
for Number_of_Array:=1 to 10 do
  If Record_Array[Number_of_Array].Class = Class then
    Write_Data;
```

Целиком текст программы, организующей ввод данных об учащихся в массив записей, а затем выводящей сведения об учащихся по номеру записи и по номеру класса, будет выглядеть следующим образом:

```
program Pupil;
type Record_Type = Record
    {Описание типа записи}
    Name      : String[10];
    SurName   : String[20];
    Age       : Byte;
    School    : Integer;
    Class     : Byte;
end;
var Record_Array : Array[1 .. 10] of Record_Type;
    Number_of_Array : 1 .. 10;
    Class           : Byte;
procedure Input_Data: {Процедура ввода данных в массив записей}
begin
  WriteLn ('Введите данные N°', Number_of_Array, ' : ');
  Write (' Ваше имя ? ');
  ReadLn (Record_Array[Number_of_Array].Name);
  Write (' Фамилия ? ');
  ReadLn (Record_Array[Number_of_Array].SurName);
  Write (' Ваш возраст ? ');
  ReadLn (Record_Array[Number_of_Array].Age);
  Write (' Школа ? ');
  ReadLn (Record_Array[Number_of_Array].Class);
  Write (' и класс ? ');
  ReadLn (Record_Array[Number_of_Array].School);
  WriteLn;
end;
procedure Write_Data: {Процедура вывода на экран содержимого текущей записи}
begin
  with Record_Array[Number_of_Array] do
  begin
    WriteLn (' Имя : ', Name);
    WriteLn (' Фамилия : ', SurName);
    WriteLn (' Возраст : ', Age);
    WriteLn (' Школа : ', School);
    WriteLn (' Класс : ', Class);
  end;
end;
```

```
{Основная программа}
begin
  for Number_of_Array:=1 to 10 do
    Input_Data;
  WriteLn;
  WriteLn ('Вывожу данные о пятом ученике : ');
  WriteLn;
  Number_of_Array:=5;
  Write_Data;
  WriteLn ('Вывожу данные по номеру класса : ');
  WriteLn;
  WriteLn (' Введите номер класса : ');
  ReadLn (Class);
  for Number_of_Array:=1 to 10 do
    If Record_Array[Number_of_Array].Class = Class then
      Write_Data;
  end.
```

Для проверки работы программы запустите интегрированную среду программирования. Введите текст программы Pupil и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Проверьте работу программы, выполнив ее в пошаговом режиме с трассировкой процедур и наблюдая в окне просмотра за изменением значений переменных Number_of_Array, Class, а также элементов массива Record_Array[Number_of_Array].

Упражнение 6. Напишем программу, создающую каталог компьютерных программ с возможностью поиска программ по фамилии автора.

Для описания сведений о программах в разделе типов введем тип Prog_Type — запись следующей структуры:

```
prog_Type = Record
    Title : string[50];
    Author : string[50];
    Entry : integer;
    Firma : String[40];
end;
```

где Title — поле для записи названия программы (строка до 50 символов);

Author — поле для записи фамилии автора (строка до 50 символов);

Entry — поле для записи года разработки (целое число);

Firma — поле для записи фирмы-разработчика (строка до 40 символов).

В разделе описания переменных введем массив Prog_Katalog из десяти записей описанного выше типа. Для указания на порядковый номер записи в массиве Prog_Katalog введем переменную Num_Array, принимающую значения от 1 до 10. Для задания шаблона поиска введем переменную строкового типа Author. Результат поиска будем записывать в переменную логического типа Yes_Prog.

Как и в предыдущей программе, используем для ввода и вывода данных специальные процедуры, поиск программы реализуем аналогично поиску учащегося, в качестве условия поиска задав условие `Prog_Katalog[Num_Array].Author = Author`. Для краткости при обращении к полям записи используем форму записи с предложением `with`.

Текст программы целиком будет выглядеть следующим образом:

```
program Kat_Prog;           {Каталог компьютерных программ}
type Prog_Type = Record
    Title : string[50];
    Author : string[50];
    Entry : integer;
    Firma : String[40];
end;
var Prog_Katalog : Array[1..10] of Prog_Type;
    Num_Array : 1..10;
    Author : string[50];
    Yes_Prog : boolean;
procedure Input_Data;      {Ввод сведений о программе}
begin
    WriteLn ('Введите данные о ', Num_Array, '-й программе :');
    with Prog_Katalog[Num_Array] do
    begin
        Write ('Название программы? ');
        ReadLn (Title);
        ReadLn (Author);
        Write ('Год разработки? ');
        ReadLn (Entry);
        Write ('Фирма ? ');
        ReadLn (Firma);
        WriteLn
    end;
end;
procedure Write_Data(Num:integer); {Вывод сведений о программе на экран}
begin
    WriteLn('Программа № ', Num);
    with Prog_Katalog[Num_Array] do
    begin
        WriteLn('Название      ', Title);
        WriteLn('Фамилия автора : ', Author);
        WriteLn('Год разработки : ', Entry);
        WriteLn('Фирма          : ', Firma);
    end;
end;
begin
    for Num_Array:=1 to 3 do      {Ввод данных о программах}
```

```
    Input_Data;
    WriteLn;
    {Поиск программ по фамилии автора}
    WriteLn('Поиск программы по фамилии автора'): WriteLn;
    Write('Введите фамилию автора : ');
    ReadLn(Author);
    Yes_Prog:=False;      {Не найдено программ этого автора}
    for Num_Array:=1 to 10 do
    if Prog_Katalog[Num_Array].Author = Author then
    begin {Если программа найдена, то напечатать сведения о ней}
        Write_Data(Num_Array); {Вызов процедуры вывода сведений о записи
с номером Num_Array на экран (Num_Array – параметр-значение)}
        Yes_Prog:=True; {Программа данного автора есть в каталоге}
    end;
    if not Yes_Prog then Write('Нет программ автора ', Author);
end.
```

Изучите текст программы `Kat_Prog`, затем запустите интегрированную среду программирования, введите текст программы и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Проверьте работу программы, выполняя ее в пошаговом режиме с трассировкой процедур и наблюдая в окне просмотра за изменением значений переменных `Num_Array`, `Author`, `Yes_Prog`, а также элементов массива `Prog_Katalog [Num_Array]`.

Контрольные вопросы и задания

Вопросы

1. Почему запись называют комбинированным типом данных?
2. Как определяется тип записи? Что называется полем записи?
3. Какие требования предъявляются к идентификаторам полей в записи?
4. Чем определяется объем памяти, требуемый для размещения записи?
5. Что такое составное имя поля записи? Из каких частей оно состоит и как записывается?
6. Зачем при обращении к полю записи используется предложение `with`?
7. Что понимается под вложенностью записей? Каков максимально допустимый уровень вложенности? Приведите примеры вложенности записей.
8. Зачем применяются записи с вариантами? Из каких частей состоит запись с вариантами?
9. Что называется полем признака? Для чего оно записывается в операторе `case`?
10. Как записываются компоненты каждого варианта записи?
11. Какие правила следует соблюдать при использовании записей с вариантами?

Задания

1. Опишите запись с именем типа *Karta*, содержащую следующие поля:
 - ☐ номер измерения (тип *integer*);
 - ☐ значение (тип *real*).
 Переменную, определяющую запись, назовите *Z*.
2. Опишите запись с именем типа *Doc*, содержащую следующие поля:
 - ☐ номер строки документа (тип *integer*);
 - ☐ текст строки (тип *string*).
 Переменную, определяющую запись, назовите *S*.
3. Опишите запись с именем типа *Tovar*, содержащую следующую информацию о хранящемся на складе товаре:
 - ☐ код товара (тип *integer*);
 - ☐ наименование товара (тип *string*);
 - ☐ цену (тип *real*).
 Переменную, определяющую запись, назовите *Tov*.
4. Опишите запись с именем типа *Graf*, содержащую данные, необходимые для построения графика из 40 точек:
 - ☐ название графика (тип *string*);
 - ☐ 40 значений (тип *integer*).
 Переменную, определяющую запись, назовите *X*.
5. Опишите запись с именем типа *Baza*, содержащую следующую информацию для школьной базы данных:
 - ☐ личный номер ученика (тип *integer*);
 - ☐ ФИО (тип *string*);
 - ☐ год рождения (тип *integer*);
 - ☐ адрес (тип *string*).
 Переменную, определяющую запись, назовите *Inf*.
6. Опишите запись с именем типа *Systema*, содержащую следующую информацию о планетах солнечной системы:
 - ☐ порядковый номер планеты (тип *integer*);
 - ☐ название планеты (тип *string*);
 - ☐ объем (*real*);
 - ☐ диаметр (*real*);
 - ☐ удаленность от Земли (*real*).
 Переменную, определяющую запись, назовите *Planeta*.

7. Опишите запись с именем типа *Sport*, содержащую следующую информацию о лучших спортивных достижениях школы по легкой атлетике:
 - ☐ название вида (тип *string*);
 - ☐ фамилия рекордсмена (тип *string*);
 - ☐ дата установления рекорда (запись *Dat*, состоящая из полей *Day*, *Month*, *Year*);
 - ☐ сообщение о результате (*real*).
 Переменную, определяющую запись, назовите *Rec*.
8. Опишите запись с именем типа *Geometr*, содержащую следующую информацию об оценках по геометрии учеников класса:
 - ☐ ФИО (тип *string*);
 - ☐ оценки за девять месяцев, не более 20 оценок в месяц.
 Переменную, определяющую запись, назовите *Dig*.
9. Опишите запись с именем типа *Rasp*, содержащую следующую информацию о движении электропоездов:
 - ☐ направление (тип *string*);
 - ☐ время отправления электропоездов (тип *real*).
 Переменную, определяющую запись, назовите *R*.
10. Опишите запись с именем типа *Post*, содержащую следующую информацию в почтовой базе данных о подписчиках на газеты и журналы:
 - ☐ ФИО (тип *string*);
 - ☐ адрес (тип *string*);
 - ☐ 10 строк с названиями газет и журналов.
 Переменную, определяющую запись, назовите *G*.
11. Опишите запись с именем типа *Boln*, содержащую следующую информацию в больничной базе данных о стационарных больных:
 - ☐ ФИО (тип *string*);
 - ☐ возраст (тип *integer*);
 - ☐ адрес (тип *string*);
 - ☐ дату поступления (тип *string*);
 - ☐ диагноз (тип *string*);
 - ☐ ФИО лечащего врача (тип *string*).
 Переменную, определяющую запись, назовите *B*.
12. Опишите запись с именем типа *Tovar*, содержащую следующую информацию о хранящемся на складе товаре:
 - ☐ код товара (тип *integer*);
 - ☐ наименование товара (тип *string*);
 - ☐ цену (тип *real*).

- Переменную, определяющую запись, назовите `Tov`. Без помощи `with` присвойте значение (10, 'туфли женские', 45200.00) полям одной из записей.
13. Опишите запись с именем типа `Data`, содержащую следующую информацию о средней температуре в хранилище за 30 дней:
- ☐ номер месяца (тип `integer`);
 - ☐ температура (тип `real`).
- Переменную, определяющую запись, назовите `Zamer`. Без помощи `with` присвойте записи начальное значение: месяц 'июль' и температура для первого дня 9,5.
14. Опишите запись с именем типа `Graf`, содержащую данные, необходимые для построения графика из 40 точек:
- ☐ название графика (тип `string`);
 - ☐ 40 значений (тип `integer`).
- Переменную, определяющую запись, назовите `X`. С помощью `with` присвойте полям записи следующие значения: название графика 'Y:=f(T)', значения первых трех точек: 5, 7, 9.
15. Опишите запись с именем типа `Post`, содержащую информацию в почтовой базе данных о подписчиках на газеты и журналы:
- ☐ ФИО (тип `string`);
 - ☐ адрес (тип `string`);
 - ☐ 10 строк с названиями газет и журналов.
- Переменную, определяющую запись, назовите `G`. С помощью `with` присвойте следующие значения полям: 'Петров И. В.', 'г. Москва, ул. Горького, 5', 'Московский комсомолец', 'Спорт'.
16. Измените программу из упражнения 5 таким образом, чтобы сведения об учащемся включали еще и адрес и был возможен поиск учащихся по фамилии.
17. Напишите программу, описывающую массив записей — телефонный справочник одноклассников — и обеспечивающую ввод данных, поиск номера телефона по фамилии, подсчет и вывод списка всех абонентов по критерию «увлечение компьютерными играми». В записи о каждом однокласснике содержатся следующие сведения: фамилия, имя, телефон, хобби.
18. Напишите программу, описывающую таблицу химических элементов, отображая следующую информацию: название, символическое обозначение, массу атома, заряд атомного ядра, список основных химических свойств. Программа должна выполнять вывод данных о химическом элементе и указанному символическому обозначению, находить элемент с самой большой массой, с самым маленьким зарядом ядра.

Напишите программу, описывающую массив записей о жильцах дома, отображая следующую информацию о каждом из них: номер квартиры, фамилия, имя, возраст, для лиц старше 18 лет в зависимости от рода занятий (учеба, работа, пенсия) — запись места учебы, места работы и трудового стажа, для пенсионеров — год выхода на пенсию. Программа должна обеспечивать ввод данных, поиск квартиры с максимальным числом жильцов, поиск самого младшего и самого старшего жильца, поиск студентов, пенсионеров.

Глава 9

Файлы

Описание файлового типа

Одной из наиболее фундаментальных структур данных, используемых в Turbo Pascal, являются файлы. Любой файл имеет три характерные особенности. Во-первых, у него есть имя, что дает возможность программе работать одновременно с несколькими файлами. Во-вторых, он содержит компоненты одного типа. Таким компонентом может быть любой тип Turbo Pascal, кроме файлового. Например, допускается файл записей или файл строк, но нельзя создать «файл файлов». В-третьих, длина создаваемого файла никак не оговаривается при его объявлении и ограничивается только емкостью устройств внешней памяти.

В большинстве случаев файлы состоят из текстовых строк, или записей. Для описания файла используется словосочетание `file of`. Синтаксическая диаграмма для файловых типов выглядит так:

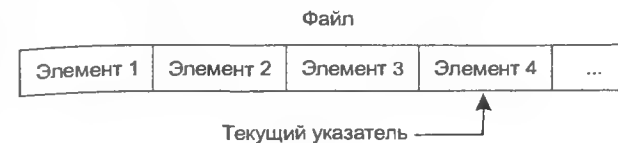


Для доступа к файлу описывается специальная файловая переменная, которая считается представителем файлов в Pascal-программе (чаще всего ее обозначают как F). Если файл состоит из записей, дополнительно описывается переменная для доступа к полям записи (обозначим ее R).

Формат:

```
type
  <имя типа> = <тип компонентов>;
var
  <F> : file of <имя типа>;
  <R> : <имя типа>;
```

Файл можно представить как потенциально бесконечный список значений одного и того же (базового) типа. Все элементы файла считаются пронумерованными, начальный элемент имеет нулевой номер.



В любой момент времени программе доступен только один элемент файла, на который ссылается текущий указатель (указатель обработки). Часто позицию размещения доступного элемента называют *текущей позицией*.

Как правило, все действия с файлом (чтение из файла, запись в файл) производятся поэлементно, причем в этих действиях участвует тот элемент файла, который обозначается текущим указателем. В результате совершения операций текущий указатель может перемещаться к тому или иному элементу файла.

По способу доступа к элементам различают файлы последовательного и прямого доступа. Файлом *последовательного доступа* называется файл, к элементам которого обеспечивается доступ в такой же последовательности, в какой они записывались. Файлом *прямого доступа* называется файл, доступ к элементам которого осуществляется по адресу элемента. Например, для поиска нужного элемента в последовательном файле необходимо, начав с нулевого элемента, перемещать указатель обработки до тех пор, пока он не будет указывать на искомый элемент, а при поиске нужного элемента в файле прямого доступа достаточно указать номер его позиции. При использовании файла последовательного доступа нельзя одновременно читать данные из файла и записывать их в файл, так как для чтения некоторого элемента последовательного файла указатель обработки перемещается к данному элементу, а для записи нового элемента этот указатель должен находиться в конце файла.

Компилятор Turbo Pascal поддерживает три типа файлов: текстовые, типизированные и нетипизированные. Прежде чем рассмотреть каждый из них в отдельности, следует познакомиться со стандартными средствами поддержки ввода-вывода в Turbo Pascal.

Средства обработки файлов

Реализованная в Pascal поддержка файловой системы наиболее полно использует возможности операционной системы по передаче данных. Каждому файлу в языке ставится в соответствие файловая переменная определенного типа, поэтому перед началом работы с файлом необходимо установить данное соответствие. Для этого в языке используется процедура Assign (var F; Name: string); где F — переменная любого файлового типа, а строковое выра-

жение Name содержит полное имя файла, отвечающее требованиям операционной системы.

Процедура Assign всегда предшествует другим процедурам работы с файлами, так как ставит в соответствие конкретному файлу на внешнем устройстве логическую файловую переменную языка, к которой впоследствии будут обращаться все другие файловые процедуры.

ВНИМАНИЕ

Недопустимо использование процедуры Assign для уже открытого файла. Это значит, что если файловой переменной с помощью процедуры Assign было назначено имя конкретного набора данных, а затем этот файл был открыт, то прежде чем использовать ту же файловую переменную для нового набора данных, необходимо с помощью процедуры Close закрыть этот файл.

Для работы с файлом прежде всего необходимо его открыть. В Pascal для этого предусмотрены две процедуры:

Reset(var F : file); – открывает существующий файл;
Rewrite(var F : file); – создает и открывает новый файл.

При описании обеих процедур параметр File обозначает файловую переменную любого типа. Открытие внешнего файла с помощью процедуры Reset в случае его отсутствия на диске может привести к ошибке при выполнении программы. Подобные ошибочные ситуации в операциях ввода-вывода позволяет отслеживать специальная функция IOresult.

Пример: Стандартное открытие файла.

```
Assign(F, '');  
Reset(F);
```

При назначении файловой переменной пустой строки происходит автоматическая ссылка на стандартный файл ввода, что в модуле SYSTEM соответствует устройству CON. С открытием такого файла появляется возможность ввода данных с клавиатуры.

Имеются некоторые различия в использовании процедуры Reset при открытии различных типов файлов. В отношении текстовых файлов (тип Text) действие процедуры означает открытие файла только для чтения. Для нетипизированных файлов в описание процедуры добавляется еще один параметр RecSize типа word, который устанавливает длину записи для функций обмена с файлом. Процедура Reset для нетипизированного файла имеет вид:

```
Reset(var F: file : RecSize: word);
```

Процедура Rewrite создает и открывает новый файл. Использование этой процедуры требует особого внимания. При попытке создать и открыть новый файл с именем уже существующего на диске набора данных действие

процедуры Rewrite сведется к удалению этого набора и созданию нового пустого файла с тем же именем.

При открытии новых нетипизированных файлов для задания длины записи в описании процедуры Rewrite добавляется дополнительный параметр RecSize типа word. В этом случае процедура имеет вид:

```
Rewrite(var F : file : RecSize: word);
```

Если процедура Rewrite используется для текстового файла, то к открываемому новому набору данных в дальнейшем могут быть применены только операции записи.

Операция закрытия файла является логическим окончанием работы с любым открытым файлом. Для этого служит процедура

```
Close(var F);
```

Использование процедуры Close позволяет устранить связь файловой переменной с внешним файлом, установленную с помощью процедуры Assign.

Пример: Полная цепочка команд для создания простого текстового файла с именем WORK.TXT:

```
var  
  F: text;  
begin  
  Assign(F, 'WORK.TXT');  
  Rewrite(F);  
  Write(F, 'Простой текстовый файл');  
  Close(F);  
end.
```

К языковым средствам обслуживания файлов необходимо отнести процедуры переименования и удаления неоткрытых файлов. Использование этих процедур не зависит от типа файла.

```
Rename(var F: NewName : string);
```

Процедура переименовывает неоткрытый файл F любого типа. Новое имя задается строкой NewName.

```
Erase(var F);
```

Процедура удаляет неоткрытый внешний файл любого типа, задаваемый переменной F.

ВНИМАНИЕ

Процедуры Rename и Erase нельзя использовать для уже открытых файлов. В противном случае могут возникнуть нежелательные последствия. Единственным стандартным шагом перед использованием процедур является установка связи между внешним файлом с конкретным именем и файловой переменной. Операции удаления и переименования осуществляются только для реально существующих файлов, иначе при выполнении программы возникает ошибка.

Пример: Удаление или переименование файла.

```
var
  F: file ;
  Ch: char;
  St: string;
begin
  Write('Введите имя файла: ');
  Readln(St);           {Чтение имени}
  Assign(F, St);        {Назначить имя файловой переменной}
  Write('Удалить файл (У), Переименовать(П), Выход(В)');
  Readln(Ch);
  case Ch of
    'У', 'у' : Erase(F);  {Удаление файла}
    'П', 'п' : begin
      Write('Введите новое имя файла: '); Readln(St);
      Rename(F, St); {Переименование файла}
    end;
    'В', 'в' : Halt(1);
  end;
end;                      {Case}
end.
```

В приведенном примере выбор тех или иных действий целиком зависит от того, что будет введено с клавиатуры. Этот вариант программы не позволяет обработать ошибочные ситуации в случае, если файл с именем St не существует на диске.

Для того чтобы файловые операции выполнялись без ошибок, необходимо использовать специальную функцию IOResult. Эта функция не имеет параметров и возвращает значение типа integer, представляющее статус последней выполненной операции ввода-вывода. Использование этой функции в программах возможно лишь в том случае, если на время выполнения файловых операций отключена стандартная проверка операций ввода-вывода. Для этих целей используется либо специальная опция в интегрированной системе, либо директива компилятора **{SI}** в тексте программы.

Пример: Вариант программы для проверки наличия файла на диске.

```
var
  F: file ;
  St: string;
begin
  Write('Введите имя файла: ');
  Readln(St);
  Assign(F, St);
  {$I-}           {Отключить стандартную обработку ошибок}
  Reset(F);       {Открыть файл}
  {$I+}           {Включить стандартную обработку ошибок}
  if IOResult = 0 then begin
    Writeln('Файл существует и нормально открыт');
    Close(F); {Закреть файл}
  end;
```

```
end
else
  Writeln('Файла с именем '+St+' на диске нет');
```

end.

После успешного выполнения операции ввода-вывода функция IOResult возвращает значение, равное нулю, в остальных случаях функция возвращает соответствующий код ошибки.

Рассмотренные операции ввода-вывода охватывают все типы файлов в Turbo Pascal и характеризуют взаимоотношения файловой и операционной систем.

ПРИМЕЧАНИЕ

Чтобы не загромождать текст программ, в большинстве примеров в данной книге отсутствуют проверки на ошибки ввода-вывода. Для их обнаружения можно использовать стандартную функцию IOResult.

Текстовые файлы

Текстовый файл можно рассматривать как последовательность символов, разбитую на строки длиной от 0 до 256 символов. Для описания используется стандартный тип Text:

```
var
  F: text;           {F — файловая переменная}
```

Каждая строка завершается маркером конца строки. На практике такой маркер представляет собой последовательность из двух символов: перевод строки chr(13) и возврат каретки chr(10). Эти два символа задают стандартные действия по управлению текстовыми файлами. Открываемые по умолчанию стандартные файлы Input и Output в модуле System имеют тип Text.

У текстовых файлов есть своя специфика. Специальные расширения стандартных процедур чтения (Read) и записи (Write) разрешают работать со значениями несимвольного типа. Другими словами, последовательность символов автоматически преобразуется к значению того типа переменной, которая используется в файловых операциях.

Вызов Read(F, Ww), где Ww — переменная типа word, осуществляет чтение из файла F последовательности цифр, которая затем интерпретируется в число, значение которого и будет присвоено переменной Ww. В случае если вместо последовательности цифр идет любая другая последовательность символов, использование такого оператора приводит к ошибке выполнения программы.

Открытие текстового файла можно произвести двумя стандартными способами:

- поставить в соответствие файловой переменной имя файла (процедура Assign), открыть новый текстовый файл (процедура Rewrite);
- поставить в соответствие файловой переменной имя файла (процедура Assign), открыть уже существующий файл (процедура Reset).

Текстовый файл в силу своей специфики во время работы допускает только один вид операции: чтение или запись. В связи с этим для работы с текстовыми файлами используется еще одна процедура открытия файла:

```
Append(var F : text);
```

Эта процедура открывает уже существующий файл и позиционирует указатель обработки на конец файла. После этого в текстовый файл можно только добавлять информацию, причем только в конец файла. На процедуру Append накладываются те же ограничения, что и на процедуры Reset и Rewrite.

Для обработки текстовых файлов используются процедуры Read и Write, обеспечивающие соответственно чтение и запись одной строки и более в текстовый файл. Использование специальных разделителей строк позволило ввести в состав языковых средств еще две процедуры: Readln, выполняющую те же действия, что и Read, и дополнительно — чтение до маркера конца строки и переход к новой строке; Writeln, обеспечивающую запись всех величин с обязательной установкой маркера конца строки в файл. Общий вид представления этих процедур следующий:

```
Readln(var F : text; V1 [.V2...Vn]):  
Writeln(var F : text; V1 [.V2...Vn]):
```

где V1...Vn — переменные разных типов.

Возникает вопрос, в каких случаях использовать Read, а когда отдать предпочтение Readln? Процедура Read обеспечивает ввод данных общим потоком из одной строки, а Readln приводит к обязательному переходу к следующей строке текстового файла, т. е. ввод данных осуществляется из различных строк. Все вышесказанное в равной мере относится к операциям записи с помощью процедур Write и Writeln.

При организации операций ввода-вывода используются специальные языковые средства в виде функций Eoln, Eof, SeekEoln, SeekEof.

Функция Eoln(var F: text) возвращает булевское значение True, если текущая файловая позиция находится на маркере конца строки или вызов Eof(F) вернул значение True. Во всех остальных случаях значение функции будет False.

Функция Eof(var F: text) возвращает булевское значение True, если указатель конца файла находится сразу за последним компонентом, и False — в противном случае.

Пример: Прочитать последовательность длиной шесть символов из первой строки текстового файла EXAMPLE.PAS.

```
var  
  F: text;  
  St: string[6];  
begin  
  Assign(F, 'EXAMPLE.PAS'); {Файл EXAMPLE.PAS должен существовать}  
  Reset(F);
```

```
while not Eoln(F) do begin {Проверка конца строки}  
  Read(F, St);  
  Writeln('St = St): {Вывод на экран}  
end;  
Readln(F); {Переход к следующей строке}  
Close(F);  
end.
```

Функция SeekEoln(var F: text) возвращает булевское значение True при достижении маркера конца строки, причем указатель файла пропускает все пробелы и знаки табуляции, предшествующие маркеру. В противном случае функция возвращает значение False.

Функция SeekEof(var F: text) возвращает значение True, если указатель файла находится на маркере конца файла. Эта функция также пропускает все пробелы и знаки табуляции, предшествующие маркеру, и выполняет автоматический пропуск маркера конца строки. Характерным примером использования этих функций может служить чтение числовых величин из текстового файла, когда необходимо пропустить обработку разделяющих эти числа пробелов или знаков табуляции.

Использование буфера ввода-вывода

Рассмотрим процедуру обмена информацией между программой и внешним текстовым файлом на диске. Для передачи данных используется буфер ввода-вывода в виде участка оперативной памяти, через который осуществляются все операции обмена. Для языка Pascal стандартный буфер ввода-вывода имеет объем 128 байт. Каждому открытому файлу вместе с обработчиком назначается и свой буфер. Возникает задача оптимизации обращений к внешним носителям информации.

Например, процедура Writeln записывает все данные последовательно в буфер. Физическая запись на внешнее устройство происходит только тогда, когда информацией будет занят последний байт буфера. После записи на диск буфер освобождается для приема следующей порции информации. Таким образом достигается компромисс между количеством и длительностью обращений к диску. Если бы каждый вызов процедуры Writeln приводил на практике к физическому обращению к диску, то тратилось бы неоправданно много времени на позиционирование головки чтения-записи для доступа к файлу.

Специально введенная процедура Flush(var F: text) дает возможность связать запись с помощью процедур Write и Writeln непосредственно с физической записью на диск. Вызов процедуры гарантирует, что все символы, переданные для записи, действительно в это же время будут записаны во внешний файл. Эту процедуру можно использовать для текстовых файлов, открытых только для записи процедурами Rewrite и Append. В прикладных программах эта

процедура используется редко, как правило, в тех случаях, когда необходимо подтверждение физической записи во внешний файл.

На практике обработка текстовых файлов сводится к считыванию всего файла в память, а затем к записи уже модифицированного файла на диск. В таких случаях с целью оптимизации времени обращения к диску целесообразно увеличение объема буфера ввода-вывода.

Процедура `SetTextBuf(var F : text; var Buf[]; size : word);` дает возможность назначить свой буфер ввода-вывода `Buf` необходимого объема `Size` текстовому файлу `F`. Параметр `Size` при вызове может быть опущен. В этом случае размер буфера соответствует `SizeOf(Buf)`. Назначение `SetTextBuf` файловой переменной `F` распространяется до следующего использования переменной новым внешним файлом.

Пример: Назначение буфера ввода-вывода.

```
var
  F: text;
  Ch: char;
  Buf: array[1..2048] of char; {Буфер 2 Кбайта}
begin
  Assign(F, ParamStr(1)); {Назначить имя файла из командной строки}
  SetTextBuf(F, Buf);      {Установить файлу буфер 2 Кбайта}
  Reset(f);
  while not Eof(F) do
    begin {Вывести файл на экран}
      Read(F, Ch);
      Write(Ch);
    end;
end.
```

Потери информации при обмене будут исключены, если переназначение буферов ввода-вывода будет осуществлено после вызова процедуры `Assign` либо сразу после процедур открытия файлов до любой операции обмена.

Типизированные файлы

К типизированным файлам относятся файлы строго определенного типа. Чаще всего это файлы, состоящие из записей. Они применяются для создания различных баз данных. Стандартное задание в программе такой файловой переменной осуществляется следующим образом:

```
type
  FileRec = record
    ...
  end;
var
  F : file of FileRec;
```

Если содержимое текстовых файлов рассматривается как набор символов, подготовленный специальным образом с учетом общепринятых соглашений о представлении текстовой информации, то содержимое типизированных файлов рассматривается как последовательность записей определенной структуры. Единицей измерения такого набора данных является сама запись. Длина записи определяется как `SizeOf(FileRec)`. Так как длина любого компонента типизированного файла строго постоянна, это дает возможность организовать прямой доступ к любому компоненту по его порядковому номеру, поэтому типизированные файлы часто называют *файлами прямого доступа*.

```
Seek(var F: NumRec: LongInt)
```

Эта процедура устанавливает текущую файловую переменную `F` на запись с номером `NumRec`; `F` — файловая переменная для типизированных и нетипизированных наборов данных. При открытии типизированного файла текущая позиция для работы с ним установлена на начало первой записи, которая по принятым соглашениям имеет номер 0, т. е. номер физической записи на единицу меньше номера логической записи. Это небольшое несоответствие в номерах может служить причиной возникновения ошибок чтения-записи, что в результате может привести к нарушению целостности важной информации. Положение усугубляется тем, что неверное позиционирование на запись с помощью процедуры `Seek`, как правило, не приводит к каким-либо видимым ошибкам ввода-вывода, на которые всегда можно отреагировать. Исключения составляют ситуации, когда нет доступа к файлу, файл не был открыт или назначено позиционирование на несуществующую запись. Такие ситуации обрабатываются с помощью функции `IOresult`.

Типизированные файлы позволяют организовать работу в режиме чтения-записи. Эта возможность играет решающую роль при определении, каким типам файлов отдать предпочтение для большинства прикладных задач. Информация в типизированных наборах данных представлена в том же виде, как и в памяти компьютера во время выполнения программы, поэтому не требуется отслеживать управляющие последовательности типа конец строки или возврат каретки.

Для работы с файлами прямого доступа дополнительно можно использовать следующие средства:

```
Truncate(var F)
```

Эта процедура уничтожает все компоненты файла `F`, начиная с места текущего положения файлового указателя.

```
FilePos(var F) LongInt
```

Эта функция возвращает для файла `F` текущую файловую позицию (номер записи, на которую она установлена) в виде значения типа `LongInt`.

```
FileSize(var F) : LongInt
```

Эта функция возвращает размер файла (количество записей) в виде значения типа LongInt.

Для пустого файла вызов FileSize возвращает значение 0. Обнаружение ошибки при обращении к внешним носителям для обеих функций производится через IOresult.

Для того чтобы очередная запись могла быть записана в конец типизированного файла, необходимо переместить текущую файловую позицию в конец файла. Когда создается новый файл, это происходит автоматически после формирования каждой очередной записи. Если файл уже создан и файловая позиция, установленная при помощи Seek, находится где-нибудь в начале файла ($\text{FilePos}(F) < \text{FileSize}(F)$), то в конец файла ее позволяет переместить вызов $\text{Seek}(F, \text{FileSize}(F))$;

В каждом файле число логических и физических записей совпадает, а при позиционировании номер физической записи на единицу меньше номера логической записи.

Когда записи располагаются в неотсортированном порядке, поиск необходимо осуществлять последовательно по всему файлу. Такое расположение записей неудобно и требует значительных расходов ресурсов системы для поиска нужной записи. Любая программная система по манипулированию базами данных всегда имеет в своем составе средства упорядочения записей по ключу.

ПРИМЕЧАНИЕ

Ключом называется совокупность знаков, идентифицирующая запись в файле. Ключ сортировки — одно или несколько полей в записи файла, по содержимому которых осуществляется упорядочение его записей, например список учащихся в классном журнале отсортирован по ключу «фамилия».

На практике это выражается в создании так называемых *индексных файлов* для главного файла данных. Индексные файлы содержат номера записей главного файла, отсортированных по конкретному ключу. Такое построение позволяет экономить время при обращении к внешним носителям, так как во всех перемещениях при сортировке участвуют записи малой длины, содержащие номера записей главного файла в соответствии с ключом сортировки.

Упражнение 1. Напишем программу, создающую на диске файл данных телефонного справочника, обеспечивающую ввод и изменение данных, а также поиск номера телефона по фамилии абонента.

В блоке описания программы определим типы:

StFio = string[20] — тип фамилия;
StPhone = string[10] — тип номер телефона.

Информацию об абонентах будем записывать в файл записями типа RecBook, в которых будут следующие поля: Fio типа StFio (в нем записывается фамилия абонента) и Phone типа StPhone (записывается номер телефона).

Для обращения к типизированному файлу опишем переменную BookFile типа file записей RecBook, для доступа к полям записи опишем переменную Work типа RecBook. Переменную Vid типа byte введем для выбора пункта меню главной программы. Переменную End_Menu типа boolean введем для организации повторения показа меню. Переменную Name типа string будем использовать для хранения значения имени файла данных справочника.

Подключим стандартный модуль Crt для использования некоторых стандартных процедур и функций (например, очистка экрана ClrScr).

Текст блока описания программы будет записан следующим образом:

```
uses Crt;
type
  StFio = string[20];
  StPhone = string[10];
  RecBook = record           {Запись сведений об абоненте}
    Fio : StFio;             {Фамилия}
    Phone : StPhone;         {Номер телефона}
  end;

var
  BookFile   file of RecBook; {Переменная для файла с записями RecBook}
  Work       RecBook;        {Переменная для доступа к записям}
  Vid : byte;
  End_Menu : boolean;
  Name : string[12];
```

Текст главной программы будет записан так:

```
begin
  ClrScr;
  End_Menu:=False;
  repeat
    {Повторять показ меню, пока End_Menu=False}
    Writeln('*** Телефонный справочник ***');
    Writeln(' Выберите вид работы:');
    Writeln(' 1 — создание нового файла');
    Writeln(' 2 — просмотр списка справочника');
    Writeln(' 3 — изменение записи');
    Writeln(' 4 — дополнение справочника');
    Writeln(' 5 — поиск абонента');
    Writeln(' 0 — завершение работы');
    Writeln(' Ваш выбор :');
    ln(Vid);
  case Vid of {Вызов разных процедур в зависимости от вида работы}
    Create_Book_Phone;
    2 OutputAllRec;
    3 UpdateRec;
    4 AddRecToEnd;
    5 FindFio;
    0 End_Menu:=True;
```

```

end;
Writeln('Для продолжения нажмите Enter');
Readln;
ClrScr;
until End_Menu: {Больше не выводить меню}
end.

```

В начале программы очищается экран, переменной `End_Menu` присваивается значение `False`, и затем на него с помощью оператора повтора `repeat` выводится текст меню из шести пунктов. После списка вариантов меню выводится запрос и считывается значение переменной `Vid`, задаваемое пользователем в соответствии с выбранным видом работы со справочником.

В зависимости от значения селектора `Vid`, оператор `case` осуществляет выбор, т. е. выбирается процедура, соответствующая значению константы выбора (например, если `Vid = 2`, то вызывается процедура `OutputAllRec` для вывода всего содержимого справочника). После выполнения процедуры управление программой передается в конец оператора `case`, и, так как значение переменной `End_Menu` не равно `True`, оператор `repeat` выполняется вновь, очищая экран и выводя список меню, и так до тех пор, пока пользователь не выберет завершение работы. В результате этого переменной `Vid` будет присвоено значение 0, из-за чего в операторе `case` переменной `End_Menu` будет присвоено значение `True`, и цикл `repeat` завершится.

В начале процедуры создания нового файла данных справочника осуществим вызов процедуры задания имени файла данных `Name_File`. Процедуру `Name_File` необходимо разместить до этого места в программе. Создание нового файла данных произведем с использованием стандартной процедуры `Rewrite`. Ввод записей об абонентах реализуем с использованием оператора повтора `for`, параметр `Ind` которого, изменяясь от 1 до числа записей `Count`, будет указывать порядковый номер записи с данными абонента в файле. Добавление одной записи в файл данных реализуем в виде процедуры `AddRec`, которую также нужно будет разместить выше в тексте программы.

В окончании процедуры организуем вывод сообщения о результатах создания файла. Для измерения размера файла в записях используем функцию `FileSize`. Текст данной процедуры можно записать так:

```

procedure Create_Book_Phone: {Создание нового файла данных}
var
  Ind, Count: integer;
begin
  Name_File;
  Assign(BookFile, Name): {Открыть новый файл для записи}
  Rewrite(BookFile);
  Writeln('Создание записей файла '.Name);
  Write('Введите число записей в справочнике');
  Readln(Count);
  for Ind := 1 to Count do

```

```

AddRec;
Writeln('Создание файла данных телефонного справочника завершено');
Writeln('Файл данных имеет '.FileSize(BookFile).' записи');
Close(BookFile);
end;

```

Процедура `Name_File`, которая вызывается для задания имени файла, будет выглядеть следующим образом:

```

procedure Name_File: {Ввод имени файла данных}
begin
  Write('Введите имя файла данных телефонного справочника >');
  Readln(Name);
end;

```

Процедура добавления одной записи в файл справочника будет выглядеть таким образом:

```

procedure AddRec: {Добавление записи в файл}
begin
  Writeln(' Ввод записи № '.FilePos(BookFile)+1); {+ 1 – указывает
                                                    номер физической записи (к номеру логической записи
                                                    добавляется 1)}
  with Work do
  begin
    Write('Введите фамилию: ');
    Readln(Fio); {Ввод фамилии}
    Write('Введите номер телефона: ');
    Readln(Phone); {Ввод номера телефона}
    Write(BookFile, Work); {Записать в файл значение переменной
                           Work: фамилию и номер телефона}
  end;
end;

```

Для сокращения записи при обращении к полям `Fio` и `Phone` переменной типа запись `Work` используем предложение `with Work do`.

Для вывода на экран всех записей файла сначала, используя функцию `IOresult`, выполним проверку наличия данного файла на диске. Если функция `IOresult` возвращает значение, отличное от 0, то на экран выводится сообщение о том, что требуемого файла на диске нет. В противном случае следует разместить указатель так, чтобы он указывал на первую запись: `Seek(BookFile, 0)`, а затем, используя оператор `while`, вызвать процедуру вывода на экран одной записи `OutputRec`. Условием окончания цикла `while` будет выражение `Eof(BookFile)`. Как только оно выполнится, цикл завершится. Данные всех записей файла выводятся на экран.

Процедуру вывода всех записей файла данных справочника запишем в следующем виде:

```

procedure OutputAllRec: {Вывод всех записей файла на экран}
begin

```



```

Name_File;
Assign(BookFile. Name);
{$I-}           {Отключить стандартную обработку ошибок}
Reset(BookFile);
{$I+}           {Включить стандартную обработку ошибок}
if IOresult = 0 then
  begin
    Seek(BookFile, 0);   {Установка на первую запись}
    WriteLn('*** Вывод телефонного справочника из файла '.Name.' ***');
    while (not Eof(BookFile)) do
      OutputRec;
    end
  else
    WriteLn('Файла с именем '+Name+' на диске нет');
end;

```

Вывод текущей записи (записи, на которой позиционирован указатель) реализуем в виде процедуры `OutputRec` с использованием предложения `with Work do`. Для вывода на экран номера записи применим функцию `FilePos(BookFile)`.

```

procedure OutputRec;      {Вывод текущей записи на экран}
begin
  Read(BookFile. Work);
  with Work do
    begin
      Write('Запись № '.FilePos(BookFile).' : ');
      WriteLn('ФИО: '.Fio.' телефон: '.Phone);
    end;
end;

```

Для изменения записи файла сначала, используя функцию `IOresult`, проверим, есть ли данный файл на диске. Если функция `IOresult` возвращает значение, отличное от 0, то на экран выводится сообщение о том, что файла на диске нет. В противном случае выводится запрос о номере изменяемой записи. После считывания номера записи переместим указатель к данной записи и, используя процедуру `OutputRec`, выведем данные этой записи из файла на экран. Задание нового значения полей этой записи осуществляется вызовом процедуры `AddRec`. В разделе описания переменных опишем локальную переменную `NumRec` типа `LongInt`, которая будет принимать значение номера изменяемой записи. Текст процедуры изменения записи файла по указанному номеру таков:

```

procedure UpdateRec;
var
  NumRec : LongInt;
begin
  Name_File;
  Assign(BookFile. Name);  {Открыть новый файл для записи}
  {$I-}

```

```

Reset(BookFile);
{$I+}
if IOresult = 0 then
  begin
    Write('Укажите номер изменяемой записи : ');
    ReadLn(NumRec);
    Seek(BookFile, NumRec-1); {Установка файловой позиции по указанному номеру}
  end;
  WriteLn('-- старое значение записи : ');
  OutputRec;      {Вывод записи и переход на следующую}
  Seek(BookFile, NumRec-1); {Возврат на прежнюю позицию}
  WriteLn('Задаем новое значение '.NumRec.' записи');
  AddRec;
  Close(BookFile);
end
else
  WriteLn('Файла с именем '+Name+' на диске нет');
end;

```

Для добавления записи в конец файла сначала, используя функцию `IOresult`, осуществим проверку наличия требуемого файла на диске. Если `IOresult` вернет ноль, то, используя стандартную процедуру `Seek(BookFile, FileSize(BookFile))`, помещаем указатель в конец файла и вызываем процедуру добавления записи `AddRec`. Затем выводим сообщение о новом размере файла и закрываем файл. Текст процедуры можно записать так:

```

procedure AddRecToEnd;
begin
  Name_File;
  Assign(BookFile. Name);  {Открыть новый файл для записи}
  {$I-}
  Reset(BookFile);
  {$I+}
  if IOresult = 0 then
    begin
      Seek(BookFile, FileSize(BookFile)); {Установка текущей позиции в конец файла}

      AddRec;
      WriteLn('Измененный файл данных имеет '.FileSize(BookFile).' записи');
      Close(BookFile);
    end
  else
    WriteLn('Файла с именем '+Name+' на диске нет');
end;

```

В процедуре поиска номера телефона опишем локальные переменные: `Maska` типа `StFio`, которая будет хранить фамилию искомого абонента; `Rez_Find` типа `boolean`, которая будет принимать значения `True` или `False` в зависимости от

результата поиска; CountRec типа integer, значение которой будет равно числу записей с такой фамилией.

После запроса имени файла данных справочника и проверки его наличия на диске сделаем запрос фамилии искомого абонента. Перед поиском присвоим переменным Rez_Find и CountRec значения False и 0 соответственно.

Просмотр всех записей файла данных при поиске реализуем при помощи оператора while. Условием завершения поиска является Eof(BookFile) — достижение конца файла. Для сокращения обращения к именам полей записи Work используем предложение with Work do. При совпадении значения поля Fio очередной записи со значением переменной Maska переменной Rez_Find присваивается значение True (абонент найден), значение переменной CountRec увеличивается на 1 и выводится сообщение о фамилии и номере телефона найденного абонента.

Текст процедуры FindFio будет выглядеть следующим образом:

```
procedure FindFio: {Поиск номера телефона по фамилии абонента}
var
  Maska: StFio;
  Rez_Find : boolean;
  CountRec: integer;
begin
  Name_File;
  Assign(BookFile, Name);
  {$I-}
  Reset(BookFile);
  {$I+}
  if IOresult = 0 then
    begin
      Write('Введите фамилию для поиска: ');
      Readln(Maska);
      Rez_Find:=False;           {Не найден абонент}
      CountRec:=0;
      while (not Eof(BookFile)) do {Просмотреть все записи до конца файла}
      begin
        Read(BookFile, Work);
        with Work do
          if Pos(Maska, Fio) <> 0 then
            begin {Найдена запись абонента с указанной фамилией}
              Rez_Find:=True;
              Inc(CountRec);
              Writeln('Фамилия: ', Fio, ' телефон: ', Phone);
            end;
          end;
      end;
      if Rez_Find then
        Writeln('Число записей для ', Maska, ' = ', CountRec)
      else
```

```
        Writeln('В справочнике нет абонентов с фамилией ', Maska);
      Close(BookFile);
    end
  else
    Writeln('Файла с именем '+Name+' на диске нет');
  end;
end;
```

Полный текст программы обслуживания телефонного справочника (создание, изменение, добавление, поиск номера телефона абонента) будет иметь следующий вид:

```
program BookPhone: {Телефонный справочник}
uses Crt;
type
  StFio = string[20];
  StPhone = string[10];
  RecBook = record {Запись сведений об абоненте}
    Fio : StFio; {Фамилия}
    Phone: StPhone; {Номер телефона}
  end;

var
  BookFile : file of RecBook; {Переменная для файла с записями RecBook}
  Work : RecBook; {Переменная для доступа к записям}
  Vid : byte;
  End_Menu : boolean;
  Name : string[12];

procedure Name_File: {Ввод имени файла данных}
begin
  Write('Введите имя файла данных телефонного справочника >');
  Readln(Name);
end;

procedure AddRec: {Добавление записи в файл}
begin
  Writeln(' Ввод записи № ', FilePos(BookFile)+1);
  with Work do
    begin
      Write('Введите фамилию: ');
      Readln(Fio); {Ввод фамилии}
      Write('Введите номер телефона :');
      Readln(Phone); {Ввод номера телефона}
      Write(BookFile, Work); {Записать в файл значение переменной Work: фамилию и номер телефона}
    end;
  end;

procedure Create_Book_Phone: {Создание нового файла данных}
var
  Ind, Count : integer;
begin
```

```

Name_File;
Assign(BookFile, Name): {Открыть новый файл для записи}
Rewrite(BookFile);
WriteLn('Создание записей файла '.Name);
Write('Введите число записей в справочнике');
ReadLn(Count);
for Ind := 1 to Count do
    AddRec;
WriteLn('Создание файла данных телефонного справочника завершено');
WriteLn('Файл данных имеет '.FileSize(BookFile).' записи');
Close(BookFile);
end;
procedure OutputRec: {Вывод текущей записи на экран}
begin
    Read(BookFile, Work);
    With Work do
        begin
            Write('Запись № ', FilePos(BookFile).' : ');
            WriteLn('ФИО: ', Fio, ' телефон: ', Phone);
        end;
end;
procedure OutputAllRec: {Вывод всех записей файла на экран}
begin
    Name_File;
    Assign(BookFile, Name);
    {$I-} {Отключить стандартную обработку ошибок}
    Reset(BookFile);
    {$I+} {Включить стандартную обработку ошибок}
    if IOresult = 0 then
        begin
            Seek(BookFile, 0); {Установка на первую запись}
            WriteLn('*** Вывод телефонного справочника из файла '.Name.' ***');
            while (not Eof(BookFile)) do
                OutputRec;
            end
        end
    else
        WriteLn('Файла с именем '+Name+' на диске нет');
end;
procedure UpdateRec;
var
    NumRec : LongInt;
begin
    Name_File;
    Assign(BookFile, Name); {Открыть новый файл для записи}
    {$I-}
    Reset(BookFile);
    {$I+}
    if IOresult = 0 then

```

```

begin
    Write('Укажите номер изменяемой записи : ');
    ReadLn(NumRec);
    Seek(BookFile, NumRec-1); {Установка файловой позиции по указанному номеру записи}
    WriteLn('-- старое значение записи : ');
    OutputRec; {Вывод записи и переход на следующую}
    Seek(BookFile, NumRec-1); {Возврат на прежнюю позицию}
    WriteLn('Задаем новое значение ', NumRec, ' записи');
    AddRec;
    Close(BookFile);
end
else
    WriteLn('Файла с именем '+Name+' на диске нет');
end;
procedure AddRecToEnd;
begin
    Name_File;
    Assign(BookFile, Name): {Открыть новый файл для записи}
    {$I-}
    Reset(BookFile);
    {$I+}
    if IOresult = 0 then
        begin
            Seek(BookFile, FileSize(BookFile)); {Установка текущей позиции в конец файла}
            AddRec;
            WriteLn('Измененный файл данных имеет '. FileSize(BookFile).' записи');
            Close(BookFile);
        end
    else
        WriteLn('Файла с именем '+Name+' на диске нет');
end;
procedure FindFio: {Поиск номера телефона по фамилии абонента}
var
    BookFile: file of RecBook;
    Work: RecBook;
    Maska: StFio;
    Rez_Find : boolean;
    CountRec : integer;
begin
    Name_File;
    Assign(BookFile, Name);
    {$I-}
    Reset(BookFile);
    {$I+}
    if IOresult = 0 then
        begin
            Write('Введите фамилию для поиска: ');
            ReadLn(Maska);

```

```

Rez_Find:=False;
CountRec:=0;
while (not Eof(BookFile)) do {Просмотреть все записи до конца файла}
begin
  Read(BookFile, Work);
  with Work do
  if Pos(Maska, Fio) <> 0 then
  begin {Найдена запись абонента с указанной фамилией}
    Rez_Find:=True;
    Inc(CountRec);
    WriteLn('Фамилия: '.Fio.' телефон: '.Phone);
  end;
end;
if Rez_Find then
  WriteLn('Число записей для '.Maska.' = '.CountRec)
else
  WriteLn('В справочнике нет абонентов с фамилией '.Maska);
Close(BookFile);
end
else
  WriteLn('Файла с именем '+Name+' на диске нет');
end;
begin {Основная программа}
  ClrScr;
  End_Menu:=False;
  repeat {Повторять показ меню, пока End_Menu=False}
  WriteLn('*** Телефонный справочник ***');
  WriteLn(' Выберите вид работы:');
  WriteLn(' 1 - создание нового файла');
  WriteLn(' 2 - просмотр списка справочника');
  WriteLn(' 3 - изменение записи');
  WriteLn(' 4 - дополнение справочника');
  WriteLn(' 5 - поиск абонента');
  WriteLn(' 0 - завершение работы');
  Write(' Ваш выбор :');
  ReadLn(Vid);
  case Vid of {Вызов разных процедур в зависимости от вида работы}
  1 : Create_Book_Phone;
  2 : OutputAllRec;
  3 : UpdateRec;
  4 : AddRecToEnd;
  5 : FindFio;
  0 : End_Menu:=True;
  end;
  WriteLn('Для продолжения нажмите Enter');
  ReadLn;
  ClrScr;
until End_Menu: {Больше не выводить меню}
end.

```

Запустите интегрированную среду программирования, введите текст программы BookPhone и запишите файл на диск под соответствующим именем, а затем откомпилируйте его и проверьте работу программы в разных режимах. Приведенный пример наглядно демонстрирует преимущества использования файлов прямого доступа. Если детализировать структуру записи ResBook, модернизировать саму программу и добавить процедуры для удаления, поиска и сортировки записей, то можно получить программу работы с базой данных.

Нетипизированные файлы

Перейдем к рассмотрению файлов, работа с которыми осуществляется с максимальной возможной скоростью. Введение таких файлов в систему Turbo Pascal было вызвано стремлением повысить эффективность программ, участвующих в интенсивном обмене с внешними наборами данных. Эти файлы, в отличие от рассмотренных выше, не имеют строго определенного типа.

Нетипизированный файл рассматривается в Pascal как совокупность символов или байтов. Представление Char или Byte не играет никакой роли, а важно лишь с точки зрения объема памяти, занимаемого данными. Такое представление стирает все различия между файлами независимо от типа их объявления. На практике это приводит к тому, что любой файл, подготовленный как текстовый или типизированный, можно открыть и начать работу с ним, как с нетипизированным набором данных. Для определения в программе нетипизированного файла служит зарезервированное слово File:

```

var
  UntypedFile : File;

```

Внутренняя реализация поддержки таких файлов наиболее близка к аппаратной поддержке работы с внешними носителями. За счет этого достигается максимальная скорость доступа к наборам данных. При работе с нетипизированными файлами не тратится время на преобразование типов и поиск управляющих последовательностей, достаточно считать содержимое файла в определенную область памяти.

Нетипизированный файл является файлом прямого доступа, что говорит о возможности одновременного использования операций чтения и записи. Для таких файлов самым важным параметром служит длина записи в байтах. Открытие нетипизированного файла с длиной записи в 1 байт можно выполнить следующим образом:

```

Rewrite(UntypedFile, 1);

```

или

```

Reset(UntypedFile, 1);

```

Второй параметр, предназначенный только для использования с нетипизированными файлами, задает длину записи файла на сеанс работы.

Особенность аппаратной поддержки заключается в том, что при обращении к внешнему устройству минимально возможным объемом данных для считывания являются 128 байт. В стремлении добиться наибольшей эффективности файловых операций в Turbo Pascal принято соглашение, по которому длина записи нетипизированного файла по умолчанию составляет 128 байт. Поэтому после открытия файла с помощью вызовов

```
Rewrite(UntypedFile);
```

или

```
Reset(UntypedFile);
```

все процедуры и функции, обслуживающие файлы прямого доступа, работают с записями длиной 128 байт. Каждый пользователь для своих программ может выбрать наиболее подходящий размер записи.

Используя для базовых операций ввода-вывода с нетипизированными файлами стандартные процедуры Read и Write, нельзя добиться большой эффективности в скорости передачи данных. Поэтому только для данного типа файлов в Turbo Pascal введены две новые процедуры, поддерживающие операции ввода-вывода с более высокой скоростью.

```
BlockRead(var F : file ; var Buf; Count : word {;Result:word});
```

Эта процедура считывает из файла F определенное число блоков в память, начиная с первого байта переменной Buf. Параметр Buf представляет любую переменную, используемую для накопления информации из файла F. Параметр Count задает число считываемых блоков. Параметр Result является необязательным, после вызова процедуры возвращается число считанных записей.

Использование параметра Result подсказывает, что число считанных блоков может быть меньше, чем задано параметром Count. Если Result указан при вызове, то ошибки ввода-вывода в такой ситуации не произойдет. Для отслеживания этой и других ошибок чтения можно использовать опции {\$I-}, {\$I+} и функцию IOresult.

```
BlockWrite(var F : file ; var Buf; Count:word {;Result:word});
```

Эта процедура предназначена для быстрой передачи в файл F определенного числа записей из переменной Buf. Все параметры процедуры BlockWrite аналогичны соответствующим параметрам процедуры BlockRead. Обе процедуры выполняют операции ввода-вывода блоками. Объем блока в байтах определяется по формуле:

Объем = Count * RecSize,

где RecSize — размер записи файла, заданный при его открытии. Суммарный объем разового обмена не должен превышать 64 Кбайт. Помимо скорости передачи данных, преимущество использования этих процедур заключается в

предоставлении программисту возможности самостоятельно определять размер буфера для файловых операций.

Эта возможность играет значительную роль в тех задачах, где необходимо жесткое планирование ресурсов.

Упражнение 2. Напишем программу, создающую нетипизированный файл из ста вещественных чисел и выводящую на экран k-й элемент файла. Проиллюстрируем обработку созданного файла двумя разными способами: поиск элемента в файле данных прямого доступа по его номеру и поиск элемента в файле данных с последовательным доступом.

В разделе описания переменных опишем файловую переменную F, представляющую в нашей программе нетипизированный файл вещественных чисел; вещественную переменную P, которой будет присваиваться значение очередного элемента файла при заполнении файла случайными вещественными числами и искомого элемента файла; целую переменную K типа byte, значения которой будут указывать на номер элемента в файле.

Раздел описания будет выглядеть так:

```
var F : file of real;
```

```
    P : real;
```

```
    K : byte;
```

Создание файла вещественных чисел реализуем в виде следующей процедуры:

```
procedure Mak_file;
```

```
begin
```

```
    Assign(F,'a.dat');
```

```
    Rewrite(F);
```

{Открыть файл для записи}

```
    Randomize;
```

```
    for K:=1 to 100 do
```

```
    begin
```

```
        P:=Random(100);
```

```
        Write(F,P); {Записать в файл значение K-го элемента}
```

```
    end;
```

```
    WriteLn('Создание файла вещественных чисел завершено');
```

```
    Close(F);
```

{Закреть файл}

```
end;
```

Поиск элемента в файле прямого доступа по его номеру представим следующим образом:

```
procedure Find_Elem: {Поиск элемента в файле прямого доступа по его номеру}
```

```
begin
```

```
    Assign(F,'a.dat');
```

```
    Write('Введите номер нужного элемента ');
```

```
    ReadLn(K);
```

```
    Reset(F);
```

{Открыть файл для чтения}

```
    Seek(F,K-1); {Переместить указатель обработки на K-1-й элемент}
```

```

Read(F,P): {Присвоить значение элемента, на который указывает указатель
обработки переменной P}
  WriteLn(K, '-й элемент файла', P:6:2);
  Close(F); {Закрыть файл}
end;

```

ПРИМЕЧАНИЕ

При позиционировании номер физической записи на единицу меньше номера логической записи, поэтому адрес элемента будет равен $K - 1$.

Поиск элемента в файле с последовательным доступом реализуем следующим образом:

```

procedure Find_Fil_P; {Поиск элемента в файле с последовательным доступом}
  var N : byte;
begin
  Assign(F, 'a.dat');
  Write('Введите номер нужного элемента ');
  ReadLn(K);
  Reset(F);
  N:=0; {Указатель обработки в начало файла}
  While not Eof(F) do {Повторять, пока не просмотрим весь файл}
  begin
    Read(F,P); {Чтение элемента и смещение указателя обработки вправо на один элемент}
    if N=K-1 then {Нашли элемент с искомым номером}
    begin
      WriteLn(K, '-й элемент файла равен ', P:6:2);
      Exit; {Прервать поиск, так как элемент найден}
    end;
    N:=N+1; {Увеличить счетчик числа элементов файла на 1}
  end;
  Close(F);
end;

```

Основной блок программы будет содержать вызовы вышеописанных процедур, а вся программа будет выглядеть следующим образом:

```

program dem_file;
uses Crt;
var F : file of real;
    P : real;
    K : byte;
procedure Mak_file; {Создание файла вещественных чисел}
begin
  Assign(F, 'a.dat');
  Rewrite(F);
  Randomize;
  for K:=1 to 100 do

```

```

begin
  P:=Random(100);
  Write(F,P);
end;
WriteLn('Создание файла вещественных чисел завершено');
Close(F);
end;
procedure Find_Elem; {Поиск элемента в файле прямого доступа по его номеру}
begin
  Assign(F, 'a.dat');
  Write('Введите номер нужного элемента ');
  ReadLn(K);
  Reset(F);
  Seek(F, K-1);
  Read(F,P);
  WriteLn(K, '-й элемент файла', P:6:2);
  Close(F);
end;
procedure Find_Fil_P; {Поиск элемента в файле последовательного доступа}
  var N : byte;
begin
  Assign(F, 'a.dat');
  Write('Введите номер нужного элемента ');
  ReadLn(K);
  Reset(F);
  N:=0; {Поместить указатель обработки в начало файла}
  While not Eof(F) do {Повторять, пока не просмотрим весь файл}
  begin
    Read(F,P); {Чтение элемента и смещение указателя обработки вправо на один элемент}
    if N=K-1 then {Нашли элемент с искомым номером}
    begin
      WriteLn(K, '-й элемент файла равен ', P:6:2);
      Exit; {Прервать поиск, так как элемент найден}
    end;
    N:=N+1; {Увеличить числа элементов файла на 1}
  end;
  Close(F);
end;
begin
  {Основная программа}
  Mak_file; {Вызов процедуры создания файла вещественных чисел}
  Find_Elem; {Вызов процедуры поиска элемента в файле прямого доступа}
  Find_Fil_P; {Вызов процедуры поиска элемента в файле с последовательным доступом}
end.

```

Запустите интегрированную среду программирования, введите текст программы `dem_file` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его и проверьте работу программы.

Обратите внимание на то, что при обработке файла a.dat как файла прямого доступа для чтения определенного элемента выполняется позиционирование указателя обработки на указанный элемент, а при обработке файла a.dat как файла последовательного доступа перемещение указателя на нужный элемент осуществляется посредством последовательного чтения элементов, начиная с первого, до тех пор, пока указатель обработки не будет установлен на искомый элемент.

Упражнение 3. Напишем программу, создающую массив целых чисел и записывающую (считывающую) его в файл разными способами (с использованием процедур Write или BlockWrite, Read или BlockRead), а также вычисляющую значение среднего арифметического всех элементов, записанных в файл.

В разделе описания программы разместим константу Count = 30000, значение которой будет определять количество элементов массива в файле Prob.dat, имя которого также описано в разделе констант. Массив из Count чисел типа byte опишем так:

```
M : array[1..Count] of byte.
```

Сумму всех элементов массива обозначим переменной Sum типа Longint. Переменную I типа integer будем использовать для указания индекса элемента массива. Целочисленная переменная Vbor будет принимать значения 1 или 2 в зависимости от выбора способа записи (чтения) файла. Переменные h, m, s, hund типа Word будут принимать значения: ч, мин, с, 0.01 с, полученные в результате вызова стандартной функции GetTime из модуля Dos.

В начале основной программы поместим создание массива M из Count случайных целых чисел.

Так как нам необходимо сравнить эффективность использования процедур чтения (записи) Write или BlockWrite, Read или BlockRead, то мы будем один и тот же массив записывать в файл и считывать из файла разными способами. Для этого в основной программе организуем выбор типа процедур 1 — Write; 2 — BlockWrite. Затем с использованием оператора case в зависимости от значения переменной Vbor реализуем вызов процедур обращения к файлу Write_File или BlockWrite_File. В каждой процедуре считаем время начала процесса создания массива и записи в файл и время окончания просмотра всех значений в файле и расчета среднего арифметического элементов. Для считывания времени используем вызов стандартной процедуры GetTime(h,m,s,hund). При выводе сообщения о текущем времени используем вызов процедуры LeadingZero, преобразующей время в строку вида: «чч:мм:сс:доли сек».

После оператора case (после окончания выполнения процедур) произведем вывод на экран значения среднего арифметического элементов файла.

Текст основной программы будет выглядеть так:

```
begin
    Randomize;
    for I:=1 to Count do
```

```
    {Основная программа}
```

```
    M[I]:=Random(200);
    WriteLn('Укажите процедуру копирования в файл данных:');
    Write('1 — Write; 2 — BlockWrite . ');
    ReadLn(Vbor);
    case Vbor of
        {Использование разных процедур записи(чтения)}
        1 : Write_File;
        2 : BlockWrite_File;
    end;
    WriteLn('Расчет окончен. Среднее значение элементов массива =', Sum/Count : 8:4);
    {Вывод вещественного числа с форматом 8:4}
    ReadLn;
end.
```

В разделе описания переменных процедуры записи (считывания) массива чисел в файл с использованием стандартных процедур Write и Read опишем файловую переменную Fil типа byte.

В начале процедуры произведем обращение к стандартной процедуре считывания системного времени GetTime, а затем выведем сообщение о начале процесса. После этого вызовем Assign и Rewrite для открытия нового файла данных. Запись элементов массива в файл реализуем с помощью оператора for, в теле которого будет использоваться процедура Write. По окончании записи всех элементов массива в файл выведем сообщение о завершении создания файла и начале считывания данных из него. Для считывания данных откроем файл для чтения и, также используя цикл for с процедурой Read, считаем все элементы, выполняя их суммирование. Как только все элементы массива будут считаны из файла, закроем файл, считаем текущее системное время и выведем его значение на экран для анализа временных затрат на процесс записи (считывания). После этого управление будет передано основной программе, в которой рассчитывается среднее арифметическое всех элементов и печатается результат.

Текст процедуры записи (считывания) массива чисел в файл с использованием стандартных процедур Write и Read будет выглядеть таким образом:

```
procedure Write_File;
var Fil file of byte;
begin
    GetTime(h,m,s,hund);
    WriteLn('Начало процесса : ', LeadingZero(h), ': ', LeadingZero(m), ': ',
    LeadingZero(s));
    Assign(Fil.FileName);
    Rewrite(Fil);
    for I:=1 to Count do
        Write(Fil,M[I]);
    Close(Fil);
    WriteLn('Файл создан. Извлекаем данные');
    Reset(Fil);
```



```

Sum:=0;
for I:=1 to Count do
begin
  Read(Fil.M[I]);
  Sum:=Sum+M[I];
end;
Close(Fil);
GetTime(h,mi,s,hund);
WriteLn('Конец процесса : '.LeadingZero(h)..''. LeadingZero(mi)..''.
LeadingZero(s));
end;

```

Текст процедуры **BlockWrite_File**, в которой создается нетипизированный файл для размещения элементов такого же массива с использованием для записи (считывания) стандартных процедур **BlockWrite** и **BlockRead**, будет почти таким же, но со следующими отличиями:

- 1) в разделе описания файловая переменная **Fil** объявляется как переменная типа **file**;
- 2) запись в файл выполняется процедурой **BlockWrite**;
- 3) чтение из файла выполняется процедурой **BlockRead**.

Далее приводится полный текст программы.

```

program demo_not_typ_file;
uses Dos, Crt;
const Count=30000;
      FileName='Prob.dat';
var
  M : array[1..Count] of byte;
  Sum : Longint;
  I : integer;
  Vyor : 1..2;
  h, mi, s, hund : Word; {Переменные ч, мин, с. 0.01 с для GetTime}
function LeadingZero(w : Word) : String; {Преобразовать время в строку
«00:00:00» с выравниванием дополнением нулями слева}
var
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  LeadingZero := s;
end;
procedure Write_File; {Создание файла типа byte с использованием Write и Read}
var Fil : file of byte;
begin
  GetTime(h,mi,s,hund);

```

```

  WriteLn('Начало процесса : '.LeadingZero(h)..''. LeadingZero(mi)..''.
LeadingZero(s));
  Assign(Fil,FileName);
  Rewrite(Fil);
  for I:=1 to Count do
    Write(Fil,M[I]);
  Close(Fil);
  WriteLn('Файл создан. Извлекаем данные');
  Reset(Fil);
  Sum:=0;
  for I:=1 to Count do
begin
  Read(Fil,M[I]);
  Sum:=Sum+M[I];
end;
  Close(Fil);
  GetTime(h,mi,s,hund);
  WriteLn('Конец процесса : '.LeadingZero(h)..''. LeadingZero(mi)..''.
LeadingZero(s));
end;
{Создание нетипизированного файла с использованием BlockWrite и BlockRead}
procedure BlockWrite_File;
var Fil : file;
begin
  GetTime(h,mi,s,hund);
  WriteLn('Начало процесса : '.LeadingZero(h)..''. LeadingZero(mi)..''.
LeadingZero(s));
  Assign(Fil,FileName);
  Rewrite(Fil);
  BlockWrite(Fil, M, 100); {Запись из массива M в файл}
  Close(Fil);
  WriteLn('Файл создан. Извлекаем данные');
  Reset(Fil);
  Sum:=0;
  BlockRead(Fil, M, 100);
  Close(Fil);
  for I:=1 to Count do
    Sum:=Sum+M[I];
  GetTime(h,mi,s,hund);
  WriteLn('Конец процесса : '.LeadingZero(h)..''. LeadingZero(mi)..''.
LeadingZero(s));
end;
begin
  Randomize;
  for I:=1 to Count do
    M[I]:=Random(200);
  WriteLn('Укажите процедуру копирования в файл данных:');

```

```

Write('1 - Write; 2 - BlockWrite : ');
Readln(Vybor);
case Vybor of
  1 : Write_File;
  2 : BlockWrite_File;
end;
WriteLn('Расчет окончен. Среднее значение элементов массива = ' Sum/Count:8:4);
Readln;
end.

```

Запустите интегрированную среду программирования, введите текст программы `demo_not_typ_file` и запишите файл на диск под соответствующим именем, а затем откомпилируйте его и проверьте работу программы, выбирая разные процедуры записи (считывания) и сравнивая время, затрачиваемое программой на обработку файла.

Упражнение 4. Рассмотрим программу, которая создает на диске файл массива целых чисел, а затем делит его на части, каждую часть записывая на диск в виде отдельного набора данных. Для использования некоторых стандартных функций подключим к программе модуль `Crt`. В блоке описания констант задается значение `Count` — размер массива целых чисел. Для ускорения ввода-вывода используется буфер `Buffer`, используемый для накопления информации, размер которого определяется константой `SizeBuffer`. Тип буфера описан как `array[1..SizeBuffer] of Byte`.

Переменные `W1`, `W2`, `W3` используются для измерения объема информации в файле (единицей измерения служит 1 буфер). Переменная `KolZap` будет принимать значения количества записей, оставшихся в делимом файле, `Siz` — размер файла-приемника (выходного файла), куда копируется часть делимого файла. Строковые переменные `NameFile` и `NamePart` будут принимать значения имен главного и выходного файла, файловые переменные `FileMain` и `FilePart` будут представлять в программе нетипизированные файлы (исходный и выходной), логическая переменная `Flag` будет принимать значения в зависимости от результата диалога о продолжении работы.

Блок описания можно записать таким образом:

```

uses Crt;
const
  Count = 15000; {Размер массива целых чисел}
  SizeBuffer = 25600; {Размер буфера для ввода-вывода}
type
  BufFile = array[1..SizeBuffer] of Byte;
var
  Buffer : BufFile;      {Буфер}
  W1, W2, W3 : word;
  KolZap, Siz : LongInt;
  NameFile, NamePart : string[60]; {Имя основного файла и выходного}
  FileMain, FilePart : file;      {Переменные под нетипизированные файлы}
  Flag : boolean;      {Флаг для логических операций}

```

Процедуру принятия решения о продолжении работы реализуем в виде функции, возвращающей `true` или `false` в зависимости от выбора пользователя. Запишем эту функцию следующим образом:

```

function Answer : boolean;
var
  Ch : char;
begin
  write('Продолжать работу' (Д/Н));
  repeat
    Readln(Ch);
  case Ch of
    'Y','y','D','d': Answer := true;
    'N','n','H','h': Answer := false;
  end;
  until Ch in ['Y','y','N','n','D','d','H','h'];
end;

```

Процедуру создания нетипизированного файла случайных целых чисел реализуем аналогично процедуре из предыдущего упражнения.

В начале основной программы вызовем процедуру создания исходного файла, затем выведем сообщение «Программа разбивает исходный файл на части», установим соответствие между файловой переменной `FileMain` и файлом, имя которого задано в переменной `NameFile`, затем откроем его для чтения и установим количество записей в нем (в байтах).

Деление исходного файла реализуем с использованием цикла `while` с условием `KolZap > 0`, т. е. повторять до конца исходного файла. В начале каждого цикла выведем сообщение о размере оставшейся части исходного файла и запрос о продолжении деления, для чего в начале тела цикла запишем:

```

Write('Введите имя выходного файла: ');
Readln(NamePart);
Assign(FilePart, NamePart);
Rewrite(FilePart, 1); {Создать выходной файл для части основного файла с
длинной записи 1}

```

Если пользователь выбирает вариант с продолжением деления, то переменной `Flag` присваивается результат выполнения функции `Answer` — `True`, и выполняется составной оператор, следующий за словом `then` в операторе условия `if Flag`. При этом запрашивается размер выходного файла, который не должен превышать число оставшихся записей в исходном файле.

Если пользователь решает не продолжать деление исходного файла, то переменной `Flag` присваивается результат выполнения функции `Answer` — `False`, и выполняется составной оператор, следующий за словом `else` в операторе условия `if Flag`: размер части задается равным размеру остатка основного файла, после чего выполняется копирование порциями, размер которых равен объему буфера. Для быстрой передачи информации из основного файла

в файл-приемник используются процедуры BlockRead и BlockWrite. Для наглядного изображения завершения копирования одной порции, равной объему буфера, используется оператор Write('+').

После этого файл-приемник закрывается, определяется размер остатка основного файла и управление передается в заголовок цикла для проверки условия KolZap > 0, и это происходит до тех пор, пока это условие выполняется. Как только оно перестанет выполняться, управление из цикла передается на оператор закрытия файла, после чего программа завершает работу.

Полный текст программы, создающей на диске файл массива целых чисел, а затем разделяющей его на части и записывающей их на диск в виде отдельных наборов данных, можно представить следующим образом:

```

Program Div_File; {Создание файла и деление его на части}
uses Crt;
const
    Count=15000;      {Размер массива целых чисел}
    SizeBuffer = 25600; {Размер буфера для ввода-вывода}
type
    BufFile = array[1..SizeBuffer] of Byte;
var
    Buffer : BufFile;      {Буфер}
    W1, W2, W3 : word;
    KolZap, Siz : LongInt;
    NameFile, NamePart : string[60]; {Имя основного файла и выходного}
    FileMain, FilePart : file; {Переменные под нетипизированные файлы}
    Flag : boolean;      {Флаг для логических операций}
function Answer : boolean;
var
    Ch : char;
begin
    Write('Продолжать работу? (Д/Н)');
    repeat
        Readln(Ch);
        case Ch of
            'Y','y','D','d': Answer := true;
            'N','n','H','h': Answer := false;
        end;
    until Ch in ['Y','y','N','n','D','d','H','h'];
end;
procedure Create_File; {Создание файла-массива случайных целых чисел}
var
    M : array[1..Count] of byte;
    I : integer;
begin
    Write('Введите имя создаваемого файла: ');
    Readln(NameFile);
    Randomize;

```

```

    for I:=1 to Count do
        M[I]:=Random(200);
    Assign(FileMain, NameFile);
    Rewrite(FileMain);      {Открыть новый файл}
    BlockWrite(FileMain, M, 100); {Записать в файл}
    Close(FileMain);
    Writeln('Создание файла окончено. нажмите Enter...');
    Readln;
end;
begin
    {Начало основной программы}
    Create_File;
    Writeln('Программа разбивает исходный файл на части');
    Assign(FileMain, NameFile);
    Reset(FileMain, 1); {Открыть основной файл, длина записи 1}
    KolZap := FileSize(FileMain); {Количество записей=размер файла в байтах}
    while KolZap > 0 do {Повторять до конца основного файла}
        begin
            Writeln('Часть файла '.NameFile.' = '.KolZap);
            Write('Деление файла. ');
            Flag := Answer; {Вызов функции диалога с пользователем}
            Writeln;
            Write('Введите имя выходного файла: ');
            Readln(NamePart);
            Assign(FilePart, NamePart);
            Rewrite(FilePart, 1); {Создать выходной файл для части основного файла
с длиной записи 1}
            if Flag then
                begin
                    Write('Введите размер выходного файла: ');
                    Readln(Siz);
                    if Siz > KolZap then {Размер выходного файла должен быть не больше
числа оставшихся записей в основном файле}
                        Siz:= KolZap;
                    end
                else
                    Siz:= KolZap; {Иначе размер части равен размеру остатка основного файла}
                    {Вводится коррекция на размер буфера ввода-вывода}
                    W1 := Siz div SizeBuffer; {Количество полных Buffer}
                    W2 := Siz mod SizeBuffer; {Остаток}
                    Write('Копирование ');
                    for W3 := 1 to W1 do {Копировать порциями=целым буферам}
                        begin
                            BlockRead(FileMain, Buffer, SizeBuffer);
                            BlockWrite(FilePart, Buffer, SizeBuffer);
                            Write('+'); {Отображать на экране завершение копирования 1 буфера}
                        end;
                    if W2 <> 0 then

```

```

begin
  BlockRead(FileMain, Buffer, W2);
  BlockWrite(FilePart, Buffer, W2);
  Write('+');
end;           {Конец оператора if W2 <> 0}
Writeln;
Close(FilePart); {Закрыть файл-приемник}
KolZap := KolZap - Siz; {Расчет остатка основного файла}
end;           {Конец оператора while KolZap > 0}
Close(FileMain);
end.

```

Запустите интегрированную среду программирования, введите текст программы Div_File и запишите файл на диск под соответствующим именем, а затем откомпилируйте его. Проверьте работу программы в пошаговом режиме без трассировки процедур, наблюдая изменение значений переменных: Flag, SizeBuffer, KolZap, W1, W2, W3, Buffer.

Если применить расширенный вызов BlockRead и BlockWrite с 4 параметрами, то можно проконтролировать внутри программы количество передаваемой информации.

Некоторые стандартные процедуры и функции обработки файлов модуля Dos

Ряд очень полезных стандартных процедур обработки файлов содержится в модуле Dos. Некоторые из них мы уже использовали ранее. Рассмотрим определенные в модуле Dos константы и переменные.

Константы

Константы флагов. Следующие константы используются для проверки отдельных битов флага в регистре Flags после вызова Intr или MSDOS:

Константа	Значение	Константа	Значение
FCarry	\$0001	FZero	\$0040
FParity	\$0004	FSign	\$0080
FAuxiliary	\$0010	FOverflow	\$0800

Например, если R — запись типа «регистр», то тест

```

R.Flags and FCarry <> 0
R.Flags and FZero = 0

```

равен True соответственно, если флаг Carry установлен и если флаг Zero сброшен.

Константы режима файла. Эти константы используются процедурами обработки файлов при открытии и закрытии дисковых файлов. Поля режимы

файловых переменных Turbo Pascal будут содержать одно из указанных ниже значений:

Константа	Значение
fmClosed	\$D7B0
fmInput	\$D7B1
fmOutput	\$D7B2
fmInOut	\$D7B3

Константы атрибутов файла. Эти константы используются для проверки, установки и очистки битов файловых атрибутов в процедурах GetFAttr, SetFAttr, FindFirst, FindNext:

Константа	Значение	Константа	Значение
ReadOnly	\$01	Directory	\$10
Hidden	\$02	Archive	\$20
SysFile	\$04	AnyFile	\$3F
VolumeID	\$08		

Эти константы можно суммировать. Например, оператор FindFirst ('*.*', ReadOnly+Directory, S); будет искать файлы «только для чтения» и подкаталоги в текущем каталоге. Константа AnyFile — это сумма всех атрибутов.

Типы файловых записей

Определения записей, используемых Turbo Pascal, также определены в модуле Dos. FileRec используется для типизированных и нетипизированных файлов, а TextRec — внутренний формат файловой переменной типа Text.

```

type
  {Типизированные и нетипизированные файлы}
  FileRec = record
    Handle      Word;
    Mode        Word;
    RecSize     Word;
    Private     : array [1..6] of Byte;
    UserData    : array [1..16] of Byte;
    Name        : array [0..79] of Char;
  end;

  {Тип записи для текстовых файлов}
  TextBuf = array [0..127] of Char;
  TextRec = record
    Handle      Word;
    Mode        Word;
    BufSize     Word;
    Private     Word;
    BufPos      Word;
    BufEnd      Word;
  end;

```

```

BufPtr   : ^TextBuf;
OpenFunc : Pointer;
InOutFunc : Pointer;
FlushFunc : Pointer;
CloseFunc : Pointer;
UserData : array [1..16] of Byte;
Name      : array [0..79] of Char;
Buffer    TextBuf;

```

end;

ПРИМЕЧАНИЕ

Запись BufPtr : ^TextBuf; означает: BufPtr — указатель на TextBuf; запись OpenFunc : Pointer; означает: OpenFunc — указатель стандартного типа Pointer. Подробнее с указателями вы познакомитесь в гл. 10.

Тип Registers. Переменные типа Registers используются процедурами Intr и MS DOS для указания входных значений содержимого регистров и проверки выходных значений содержимого регистров процессора для программного прерывания.

```

type
  Registers = record
    case Integer of
      0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Word);
      1: (AL, AH, BL, BH, CL, CH, DL, DH: Byte);
    end;

```

Заметим, что можно пользоваться одновременно 8-разрядными и 16-разрядными регистрами.

Тип DateTime. Переменные типа DateTime используются в сочетании с процедурами UnpackTime и PackTime для проверки и создания 4-байтных упакованных значений даты и времени в процедурах GetFTime, SetFTime, FindFirst и FindNext:

```

type
  DateTime = record
    Year, Month, Day, Hour, Min, Sec: Integer;
  end;

```

Диапазон допустимых значений: Year 1980..2099, Month 1..12, Day 1..31, Hour 0..23, Min 0..59, Sec 0..59.

Тип SearchRec. Переменная типа SearchRec используется процедурами FindFirst и FindNext для просмотра справочников:

```

type
  SearchRec = record
    Fill: array[1..21] of Byte;
    Attr: Byte;
    Time, Size: Longint;

```

```

Name: String[12];
end;

```

Информация, найденная для каждого файла одной из этих процедур, возвращается в SearchRec. Поле Attr содержит атрибуты файла (сформировано из констант атрибутов), Time содержит упакованные время и дату (используйте UnpackTime для распаковки), Size содержит размер файла в байтах и Name содержит имя файла. Поле Fill резервируется операционной системой и никогда не должно модифицироваться.

Строковые типы обработки файлов. Эти строковые типы используются в процедуре FSplit:

```

DirStr = String[67]; {Строка устройства и справочника}
NameStr = String[8]; {Строка имени файла}
ExtStr = String[4]; {Строка расширения файла}
ComStr = String[127]; {Командная строка}
PathStr = String[79]; {Полная строка пути файла}

```

Переменные

Переменная DosError. Переменная DosError используется многими программами в модуле Dos для указания ошибок.

```
var DosError Integer;
```

Значение, возвращаемое в DosError, представляет собой код ошибки операционной системы. Значение 0 означает «нет ошибки», другие коды означают:

Код ошибки DOS	Значение
2	Файл не найден
3	Путь не найден
5	Доступ запрещен
6	Неверный обработчик
8	Нет памяти
10	Неправильная среда
11	Неправильный формат
18	Больше нет файлов

Процедуры и функции

Процедуры даты и времени

Процедура	Описание
GetDate	Возвращает текущую дату, установленную в DOS
GetFTime	Возвращает дату и время последней записи в файл
GetTime	Возвращает текущее время, установленное в DOS

(продолжение)

Процедура	Описание
PackTime	Преобразует запись в 4-байтное упакованное значение даты и времени типа longint, используемое процедурой SetFTime. Поля записи DateTime не проверяются на диапазон
SetData	Устанавливает текущую дату в DOS
SetFTime	Устанавливает время и дату последней записи в файл
SetTime	Устанавливает текущее время в DOS
UnpackTime	Преобразует 4-байтное упакованное значение даты и времени, возвращаемое GetFTime, FindFirst или FindNext в распакованную запись типа DateTime

Функция статуса диска

Функция	Описание
DiskFree	Возвращает число свободных байтов на указанном диске
DiskSize	Возвращает полный объем указанного диска в байтах

Процедуры обслуживания прерываний

Процедура	Описание
GetIntVec	Возвращает адрес, хранящийся в указанном векторе прерывания
Intr	Выполняет указанное программное прерывание
MSDos	Выполняет функцию операционной системы
SetIntVec	Устанавливает адрес для указанного вектора прерывания

Процедуры обработки файлов

Процедура	Описание
FindFirst	Ищет в указанном или текущем каталоге первый файл, соответствующий заданному имени файла и набору атрибутов
FindNext	Возвращает следующий файл, соответствующий имени и атрибутам, указанным в предыдущем вызове FindFirst
GetFAttr	Возвращает атрибуты файла
SetFAttr	Устанавливает атрибуты файла
FSplit	Разбивает имя файла на три составные части (каталог, имя файла, расширение)

Функции обработки файла

Функция	Описание
FExpand	На основе имени файла возвращает полное имя файла (устройство, каталог, имя и расширение)
FSearch	Ищет файл в списке каталогов

Процедуры обработки процессов

Процедура	Описание
Exec	Выполняет заданную программу с указанной командной строкой
Keep	Завершает программу и оставляет ее в памяти (реализует прерывание TSR, «завершить и оставить резидентным»)
SwapVectors	Меняет местами сохраненные векторы прерываний с текущими векторами

Функции обработки процессов

Функция	Описание
DosExitCode	Возвращает код завершения подпроцесса

Функции управления средой

Функция	Описание
EnvCount	Возвращает число строк среды
EnvStr	Возвращает указанную строку среды
GetEnv	Возвращает значение указанной переменной среды

Дополнительные функции

Функция	Описание
DosVersion	Возвращает номер версии DOS

Дополнительные процедуры

Процедура	Описание
GetCBreak	Возвращает состояние проверки Ctrl+Break в DOS
SetCBreak	Устанавливает состояние проверки Ctrl+Break в DOS
GetVerify	Возвращает состояние флага верификации в DOS
SetVerify	Устанавливает состояние флага верификации в DOS

Упражнение 5. Написать программу, производящую сканирование диска, определение его размера, объема свободного пространства, вывод списка файлов по заданному шаблону, поиск указанного файла и его переименование.

Для использования стандартной функции очистки экрана подключим к программе модуль Crt, а для использования стандартных процедур обработки файлов — модуль Dos. В разделе описания переменных определим файловую переменную Fil, представляющую в программе файл данных типа byte, переменную FileInfo типа SearchRec, в которой стандартными процедурами FindFirst и FindNext при просмотре каталогов записывается следующая инфор-

мация: имя и расширение файла, размер в байтах, упакованные дата и время создания, атрибут.

Целая переменная `Ftime` будет содержать упакованные время и дату создания файла, считанные процедурой `GetFTime`. В переменной `T` типа `DateTime` будем хранить распакованные время и дату. Для отображения распакованного значения времени создания файла введем целые переменные `H`, `M`, `S`, `Hund`, значения которых будут содержать, соответственно: часы, минуты, секунды, сотые доли секунд. Строковые переменные `Dir_s`, `N`, `E`, `Name`, `New_Name` будут использоваться для хранения значений каталога, имени, расширения файла, старого и нового имени файла при переименовании. Целая переменная `I` будет принимать значения количества найденных в каталоге файлов, а `P` — указывать на точку, разделяющую имя и расширение файла.

Для преобразования распакованного значения времени в строку текста будет использоваться функция `LeadingZero`, которая преобразует полученное из основной программы целочисленное значение (ч, мин, с) в строковый тип, при длине полученной строки 1 добавляет в начало строки ноль и возвращает результат основной программе. Текст подпрограммы-функции `LeadingZero` будет таким:

```
function LeadingZero(W : Word) : string;
var
  S : string;
begin
  Str(W:0.S);
  if Length(S) = 1 then
    S := '0' + S;
  LeadingZero := S;
end;
```

Для отделения имени файла от расширения мы будем использовать процедуру `Nam_Ext_File`. В этой процедуре при помощи стандартной функции `Pos` определяется позиция точки, отделяющей имя от расширения, а затем производится копирование значения части считанного из каталога имени файла (поля `FileInfo.Name`) от первого символа до позиции точки в переменную `N`, а значения части имени файла от позиции, следующей за позицией точки, до конца имени (расширение имени файла) — в переменную `E`. Если в строке `FileInfo.Name` нет точки, то все значение `FileInfo.Name` присваивается переменной `N`. Текст этой процедуры выглядит следующим образом:

```
procedure Nam_Ext_File; {Отделить имя и расширение файла}
begin
  P := Pos('.', FileInfo.Name); {Определить позицию символа «.» в имени файла}
  if P > 1 then
    begin
      N := Copy(FileInfo.Name, 1, P - 1);
      E := Copy(FileInfo.Name, P + 1, 3);
    end
```

```
else {Расширение отсутствует}
begin
  N := FileInfo.Name;
  E :=
end;
end;
```

Процедуру переименования указанного файла реализуем с использованием стандартной процедуры `Rename`, дополнив ее запросом и вводом нового имени файла и выводом на экран сообщения о результатах переименования.

В начале основной программы очистим экран компьютера, при помощи стандартной функции `GetTime` считаем системное время и выведем сообщение о текущем времени. Затем при помощи стандартной процедуры `GetDir(0, Dir_s)` считаем в переменную `Dir_s` каталог текущего диска (параметр 0 указывает текущий дисковод) и напечатаем значение переменной `Dir_s`. Используя стандартные процедуры `DiskSize` и `DiskFree`, реализуем вывод на экран сообщения о размере текущего диска и свободном пространстве на диске.

Перед началом поиска присвоим переменной `I` значение 0 и вызовем процедуру `FindFirst('*.pas', Archive, FileInfo)` для поиска в текущем каталоге файлов с расширением `.PAS` и атрибутом «архивный». После вывода на экран заголовка списка файлов «Имя файла Размер(байт) Дата Время создания» реализуем поиск файлов при помощи оператора `while`. Поиск будет повторяться до тех пор, пока выполняется условие `DosError = 0`.

В начале тела цикла увеличивается счетчик числа найденных файлов заданного шаблона, затем файловая переменная связывается с файлом, имя которого определяется значением поля `Name` записи `FileInfo` типа `SearchRec`. Затем вызывается процедура отделения имени файла от расширения, после чего на экран выводится значение имени и расширения файла, а также значение поля `Size` записи `FileInfo` типа `SearchRec`, которое указывает размер файла в байтах.

Стандартная функция `GetFTime` возвращает дату и время последнего изменения файла в упакованном виде. Для распаковки применим процедуру `UnpackTime`. После этого с использованием точечной нотации запишем вывод на экран значений полей даты и времени записи `T` типа `DateTime`.

Для продолжения поиска файлов, начатого процедурой `FindFirst` с прежним именем и атрибутами, используем вызов процедуры `FindNext`, после чего управление передается в заголовок цикла для проверки условия `DosError = 0` и если оно выполняется, операторы тела цикла выполняются еще раз.

Если условие `DosError = 0` не выполняется, то управление передается за пределы цикла поиска файлов, и на экран выводится сообщение о количестве найденных файлов.

Переименование файла осуществляется с использованием оператора `repeat`. Для поиска файла с заданным именем используем функцию поиска `FSearch`.

Если функция возвращает пустую строку, то в каталогах, описанных в переменной окружения PATH, нет файлов. Для считывания переменной окружения, в которой записан маршрут поиска файлов, используем функцию GetEnv. Условием окончания поиска файла является выражение FSearch(Name, GetEnv('PATH')) <> '', т. е. файл для переименования успешно найден.

После этого при помощи процедуры Assign свяжем файловую переменную Fil с указанным файлом и вызовем процедуру Rename_File для переименования файла.

Полный текст программы можно записать следующим образом:

```
program Find_Ren_File;
uses Dos, Crt;
var
  Fil : file of byte;
  FileInfo : SearchRec;
  H, M, S, Hund : word; {Переменные ч. мин. с. 0.01 с для GetTime}
  Ftime : longint; {Упакованные время и дата в GetFTime}
  T : DateTime; {Распакованные время и дата в переменной типа DateTime}
  Dir_s, N, E, Name, New_Name : string;
  I, P : integer;
function LeadingZero(W : Word) : string; {Преобразовать время в строку}
var
  S : string;
begin
  Str(W:0,S);
  if Length(S) = 1 then
    S := '0' + S;
  LeadingZero := S;
end;
procedure Nam_Ext_File; {Отделить имя и расширение файла}
begin
  P := Pos('.', FileInfo.Name); {Определить позицию символа «.» в имени файла}
  if P > 1 then
    begin
      N := Copy(FileInfo.Name, 1, P - 1);
      E := Copy(FileInfo.Name, P + 1, 3);
    end else
      {Расширение отсутствует}
    begin
      N := FileInfo.Name;
      E := '';
    end;
end;
end;
procedure Rename_File; {Переименовать файл}
begin
  Write('Введите новое имя файла: ');
  Readln(New_Name);
```

```
  Rename(Fil, New_Name);
  Writeln('Файл ' . Name . ' переименован в ' . New_Name . ', нажмите Enter...');
  Readln;
end;
begin
  {Основная программа}
  ClrScr;
  GetTime(H, M, S, Hund); {Прочитать системное время}
  Writeln('Текущее время ' . LeadingZero(H) . ' ' . LeadingZero(M) . ' ' .
    LeadingZero(S));
  GetDir(0, Dir_s); {0 = читать каталог текущего диска}
  Writeln('Текущий диск и каталог ' . Dir_s);
  Writeln(DiskSize(0) div 1024, ' Кбайт всего на диске');
  Writeln(DiskFree(0) div 1024, ' Кбайт свободно');
  Writeln;
  I := 0; {Пока не найдено ни одного файла}
  FindFirst('*.*', Archive, FileInfo);
  Writeln('Имя файла Размер(байт) Дата Время создания');
  Writeln;
  while DosError = 0 do {Пока поиск файла завершается успешно}
    begin
      I := I + 1; {Найден еще один файл}
      Assign(Fil, FileInfo.Name);
      Nam_Ext_File; {Отделить имя файла от расширения}
      Write(N, ' ' . 9 - Length(N), E, ' ' . 4 - Length(E));
      Write(' ' . FileInfo.Size: B);
      GetFTime(Fil, Ftime); {Возвратить дату и время последнего изменения файла}
      UnpackTime(FileInfo.Time, T); {Преобразует 4-байтовые, упакованные в Longint
время и дату в распакованную запись типа DateTime}
      Write(' ' . T.Day:2, '-', T.Month:2, '-', T.Year:4);
      Writeln(' ' . T.Hour:2, ':', T.Min:2, ':', T.Sec:2);
      FindNext(FileInfo);
    end;
  Writeln;
  Writeln('В текущем каталоге найдено ' . I . ' архивных файлов с расширением
.PAS');
  repeat
    Write('Введите имя файла для переименования >');
    Readln(Name);
  until Name in ['.', '']; {Искать файл в каталогах, описанных в переменной окружения PATH}
  if FSearch(Name, GetEnv('PATH')) = '' then
    Writeln('На диске нет файла ' . Name);
  until FSearch(Name, GetEnv('PATH')) << @062 > ''; {Найден файл Name}
  Assign(Fil, Name);
  Rename_File; {Вызов процедуры переименования файла}
end;
```

Запустите интегрированную среду программирования, введите текст программы Find_Ren_File и запишите файл на диск под соответствующим име-

нем, а затем откомпилируйте его. Проверьте работу программы в пошаговом режиме с трассировкой процедур, наблюдая за изменением значений переменных: `H`, `M`, `S`, `Hund`, `Dir_s`, `FileInfo.Name`, `N`, `E`, `FileInfo.Size`, `T.Year`, `T.Month`, `T.Day`, `T.Hour`, `T.Min`, `T.Sec`.

Контрольные вопросы и задания

Вопросы

1. Назовите общие и отличительные черты текстовых, типизированных и нетипизированных файлов.
2. Для чего используется специальная файловая переменная? Как устанавливается соответствие файловой переменной файлу во внешней памяти?
3. Что общего у процедур `Reset` и `Rewrite` и чем они отличаются?
4. Какие отличия существуют в использовании процедуры `Reset` при открытии различных типов файлов (текстовых, нетипизированных)?
5. Зачем применяется процедура `Close`?
6. Какие процедуры применяются для переименования и удаления файлов? Каковы особенности их использования?
7. Для каких целей используется специальная функция `IOresult`? Каковы условия ее применения? Каково назначение директивы компилятора `{SI}`? Что возвращает функция `IOresult` после корректного выполнения операции ввода-вывода?
8. В чем заключается специфика текстовых файлов? Каково назначение процедуры `Append`? Опишите отличительные особенности процедур `Read` и `Write` от `Readln` и `Writeln`.
9. Объясните назначение функций `Eoln`, `Eof`, `SeekEoln`, `SeekEof`.
10. Объясните назначение процедуры `Flush` и особенности ее использования.
11. Объясните назначение процедуры `SetTextBuf`, расскажите о ее параметрах.
12. Какие файлы относятся к типизированным? Как представлена информация в типизированных файлах?
13. Опишите назначение процедур `SizeOf`, `Seek`, `Truncate`, `FilePos`, `FileSize`.
14. В чем заключается несоответствие номера физической записи и номера логической записи в типизированном файле?
15. Какие файлы называются нетипизированными? Как они определяются, каковы их особенности?
16. Каково назначение процедур `BlockRead` и `BlockWrite`?
17. Назовите стандартные процедуры обработки файлов, содержащиеся в модуле `Dos`. Каково назначение переменной `DosError`?

18. Что общего и каковы отличия процедур обработки файлов `FindFirst` и `FindNext`, функций `FSearch` и `FExpand`?
19. Можно ли, считав из файла пятый элемент, затем сразу же считать второй элемент? А какой можно?
20. В какое место файла можно добавлять новые элементы: в начало, в конец, в середину, куда угодно, никуда?
21. Если не переписывать файл заново, то значения каких элементов можно изменять: только первого, только последнего, каких угодно, никаких? Какие элементы можно удалять из файла (при том же условии)?
22. Верно ли, что в одно и то же время нельзя считывать из файла и записывать в него? Верно ли, что начав считывать из файла, затем уже никогда нельзя записывать в него? А наоборот?

Задания

1. Напишите программу, создающую файл, состоящий из 10 значений типа `integer`. Прочитайте файл и вычислите сумму его элементов. Тип `record` не используйте.
2. Напишите программу, создающую файл, состоящий из неопределенного количества значений типа `integer`. Для ввода используйте цикл, выход из цикла — значение 999. После записи выведите файл на экран и уничтожьте файл. Тип `record` не используйте.
3. Напишите программу, создающую файл, состоящий из 10 значений типа `integer`. Прочитайте файл и определите, есть ли в нем введенное с клавиатуры значение. Тип `record` не используйте.
4. Напишите программу, которая создает файл из элементов типа `Char` с помощью цикла `while`. Признак выхода из цикла — буква 'z'. Скопируйте созданный файл в другой файл и выведите его на экран.
5. Напишите программу, создающую файл, состоящий из пяти значений типа `real`. Тип `record` не используйте. Выведите файл на экран. В цикле `while..do` расширьте файл за счет добавления новых значений. Выход из цикла — 999. После расширения выведите файл на экран.
6. Напишите программу, создающую файл, состоящий из `N` значений типа `integer`. Прочитайте файл и выведите только четные элементы. Тип `record` не используйте.
7. Создайте файл, компоненты которого являются целыми числами. Напишите программу, перезаписывающую компоненты файла в обратном порядке (без создания нового файла).
8. Дан файл `f`, компоненты которого являются целыми числами. Напишите программу, записывающую в файл `q` все компоненты файла `f`, делящиеся на 5 и принадлежащие интервалу `[C,D]`.

9. Напишите программу, которая создает файл, компоненты которого имеют следующую структуру:

- ☐ табельный номер;
- ☐ ФИО;
- ☐ оклад.

Введите в файл данные о пяти работниках, выведите в другой файл данные о работнике, имеющем максимальный оклад.

10. Напишите программу, создающую файл из элементов типа `Char` с помощью цикла `while`. Признак выхода из цикла — буква 'z'. Присвойте файлу новое имя 'D2.DOC' и выведите его содержимое на экран.

11. Напишите программу, создающую и выводящую на экран файл 'ZARPL.DAT', компоненты которого имеют следующую структуру:

- ☐ табельный номер;
- ☐ ФИО;
- ☐ сумма зарплаты.

Выход из ввода — табельный номер, равный 999. Выведите на экран табельные номера, ФИО и зарплату тех работников, у которых зарплата превышает 100000.00 руб. Используйте оператор `with`.

12. Напишите программу, создающую и выводящую на экран файл 'AVANS.DAT', компоненты которого имеют следующую структуру:

- ☐ табельный номер;
- ☐ аванс.

Выход из ввода — табельный номер, равный 999. Получите ведомость следующей структуры:

Таб. номер	Аванс
1	100.00
2	200.00
Итого:	300.00

13. Напишите программу, которая создает файл 'RANDOM1.DAT', состоящий из 50 случайных чисел типа `integer` в диапазоне 0..200. После создания выведите элементы файла на экран.

14. Напишите программу, создающую файл 'RANDOM2.DAT', состоящий из 100 случайных чисел типа `integer` в диапазоне 0..300. Исследуйте получившийся файл с целью обнаружения в нем простых чисел 23, 31, 37, 41, 53, 107, 127, 151, 197. В конце программы уничтожьте созданный файл.

15. Напишите программу, создающую файл 'F1.DTA' из 10 элементов типа `integer`. Выведите его на экран. Удалите последние пять элементов и выведите содержимое файла на экран.

16. Напишите программу, создающую файл 'OLD.T' из элементов типа `char` с помощью цикла `repeat`. Признак выхода из цикла — символ '!'. Присвойте файлу новое имя 'NEW.T' и выведите его содержимое на экран.

17. Напишите программу, создающую файл из 20 компонентов: 1, 2, ..., 20 типа `integer` с помощью `for` без ввода с клавиатуры. Выведите файл на экран. Дайте компоненту номер 15 новое значение — 99 — и снова выведите файл на экран, затем уничтожьте файл.

18. Напишите программу, считывающую текст из файла, заменяющую в нем все буквы «о» на «а» и записывающую файл на диск.

19. Напишите программу, создающую файл с записями о владельцах автомобилей: марка автомобиля, номер регистрации в ГИБДД, дата постановки на учет, ФИО владельца, домашний адрес (область, город, район, улица, дом, квартира) и обеспечивающую обслуживание данного файла: запись, изменение и удаление данных, а также поиск данных по регистрационному номеру.

20. Напишите программу, считывающую с диска файл, в котором записана некоторая последовательность символов, и переписывающую эти символы в другой файл, исключая символы, расположенные между скобками (.). Сами скобки тоже исключаются. Предполагается, что внутри каждой пары скобок нет других скобок.

21. Напишите программу, построчно выводящую содержимое текстового файла на экран и на печать.

22. Имеется текстовый файл. Напишите программу переформатирования этого файла с разбиением на строки так, чтобы каждая строка оканчивалась точкой или содержала ровно 60 символов, если среди них нет точки.

23. Имеется файл из целых чисел. Напишите программу упорядочения файла по убыванию.

24. Напишите программу, создающую файл таблицы значений функций $\sin(x)$ и $\operatorname{Tg}(x)$ на отрезке $[0,3]$ с шагом 0.01. Значения x следует записывать с одной цифрой в дробной части, значения функции $\sin(x)$ — с пятью, а значения $\operatorname{Tg}(x)$ — в экспоненциальной форме.

25. Дан файл D, содержащий даты. Каждая дата — это число, месяц и год. Следует найти и записать в файл D1 год с наименьшим номером и самую позднюю дату, а в файл D2 — все весенние даты.

26. Напишите программу, создающую файл, в котором содержатся сведения об игрушках: указывается название игрушки, ее стоимость и возрастные

границы (например, игрушка может предназначаться для детей от двух до пяти лет). Программа должна выполнять изменение данных и поиск по следующим критериям:

- 1) названия игрушек, цена которых не превышает 4000 р. и которые подходят детям до четырех лет;
 - 2) цена и название самой дорогой игрушки;
 - 3) названия игрушек, которые подходят детям как четырех лет, так и десяти лет.
27. Напишите программу, создающую на диске файл с ведомостью успеваемости учащихся класса, в котором будет записана следующая информация: фамилии, имена учащихся, название предмета, оценки за две контрольные работы по трем предметам. Программа должна обеспечивать ввод, редактирование данных и вывод списков учащихся:
- 1) выполнивших первую работу по всем предметам на 5;
 - 2) выполнивших хотя бы одну работу на 5;
 - 3) выполнивших обе работы хотя бы по одному предмету на 5;
 - 4) выполнивших все работы на 4 и 5;
 - 5) выполнивших две работы на 4 и 5;
 - 6) выполнивших все работы на 3.
- Выведите на экран: число учеников, выполнивших обе работы на 4; не выполнивших ни одной работы.
28. Напишите программу, записывающую в файл закодированный текст, считывающую его и выполняющую дешифрование, если известен код шифрования — число, указывающее смещение букв в алфавите (например, код 3 означает, что вместо буквы «а» в зашифрованном тексте стоит буква «в»). Дешифрованный текст следует записать в другой файл.
29. Напишите программу, записывающую в файл одномерный массив случайных целых чисел, а затем считывающую его с диска и выполняющую запись четных элементов массива в другой файл.
30. Напишите программу, создающую файл записей — телефонный справочник одноклассников. Программа должна обеспечивать ввод данных, поиск номера телефона по фамилии, подсчет и вывод списка всех абонентов по критерию «увлечение компьютерными играми». В записи о каждом однокласснике содержатся следующие сведения: фамилия, имя, телефон, хобби.
31. Напишите программу, создающую файл данных о химических элементах, отображая следующую информацию: название, символическое обозначение, массу атома, заряд атомного ядра, список основных химических свойств. Программа должна выполнять вывод данных о химическом эле-

менте по указанному символическому обозначению, находить элемент с самой большой массой, с самым маленьким зарядом ядра.

32. Напишите программу, создающую файл данных о жильцах дома, отображая в нем следующую информацию о каждом: номер квартиры, фамилию, имя, возраст, для лиц старше 18 лет в зависимости от рода занятий (учеба, работа, пенсия) — запись места учебы, места работы и трудового стажа, для пенсионеров — год выхода на пенсию. Программа должна обеспечивать ввод данных, поиск квартиры с максимальным числом жильцов, поиск самого молодого и самого пожилого жильца, поиск студентов, пенсионеров.

Глава 10

Динамические структуры данных

В любой вычислительной системе одним из важнейших ресурсов является память. Управление памятью — одна из главных задач, стоящих перед программистом. Очень важно создавать программы, эффективно использующие память, так как во время выполнения программы память необходима для следующих элементов программ и данных:

- сама программа пользователя;
- системные программы этапа выполнения, осуществляющие вспомогательные действия при работе программы пользователя;
- определяемые пользователем структуры данных и константы;
- точки возврата для подпрограмм;
- временная память для хранения промежуточных результатов при вычислении выражений;
- временная память при передаче параметров;
- буферы ввода-вывода, используемые как временные области памяти, в которых хранятся данные между моментом их реальной физической передачи с внешнего устройства или на него и моментом начала операции ввода или вывода в программе;
- различные системные данные (информация о статусе устройств ввода-вывода и т. п.).

Из этого списка видно, что управление памятью касается широкого класса объектов. Ранее мы использовали простейший способ распределения памяти — статическое распределение, т. е. распределение памяти при трансляции программы. Статическое распределение памяти эффективно, поскольку во время выполнения программы на управление памятью не расходуется ни время, ни память. Далее вы познакомитесь с динамическим управлением памятью, осуществляемым во время выполнения программы.

Статические и динамические переменные

Практика программирования часто выдвигает на передний план два конфликтующих фактора: время выполнения программы и объем занимаемо

памяти. Конечно, высокое быстродействие программы всегда желательно, но бывают случаи, когда важнее обеспечить максимальную экономию памяти даже ценой потери в скоростных характеристиках. Рассмотрим пример. Пусть в программе обрабатывается матрица 300×300 целых чисел, тогда ее нужно описать следующим образом:

```
var M1 : array[1..300,1..300] of integer;
```

Такие переменные, описанные в разделе `var`, Н. Вирт назвал *статическими*. Такое название они получили за то, что компилятор Pascal может обработать их без выполнения программы, на основании лишь статического текста программы. Статические переменные можно использовать в случаях, когда необходимый для работы программы объем памяти известен в момент написания программы.

В данном случае мы имеем наглядный пример нерационального использования памяти с применением статических переменных. Так как один элемент матрицы — целое число — занимает в памяти два байта, а общее количество элементов составляет $300 \times 300 = 90\,000$, то для размещения всей матрицы вышеописанным способом в памяти потребуется $90\,000 \times 2 = 180\,000$ байт. Вместе с тем маловероятно, чтобы при каждом выполнении программы ей действительно потребовались одновременно все элементы такого огромного массива. Более рациональным подходом является использование динамической памяти. *Динамическая память* — это оперативная память, предоставляемая программе при ее работе, за вычетом сегмента данных, стека и собственно тела программы.

Решение проблемы экономного расходования памяти состоит в том, чтобы не резервировать заранее максимальный объем памяти для размещения данных, а предварительно определив тип данных, создавать новый экземпляр данных всякий раз, когда в нем возникает необходимость. Такие переменные, которые создаются и уничтожаются в процессе выполнения программы, называются *динамическими*.

Помимо экономии памяти, использование динамических переменных обусловлено существованием задач, для которых невозможно предсказать объем требуемой памяти в момент написания программы, и память должна выделяться динамически в процессе работы. Динамическая память широко используется для временного хранения данных при работе с графическими и звуковыми средствами компьютера.

Указатели

Все рассмотренные ранее типы данных содержат непосредственно данные, размещенные в памяти компьютера. Для организации динамической памяти применяются особые переменные, называемые *указателями*. Их назначение состоит в том, чтобы указывать местоположение какой-то другой перемен-

ной заранее определенного типа. Таким образом, указатель — это переменная, которая в качестве своего значения содержит адрес первого байта памяти, по которому записаны данные. Сам по себе указатель занимает в памяти всего четыре байта, а данные, на которые он указывает, могут занимать десятки килобайт. Переменной, на которую ссылается указатель, не обязательно присваивать какое-то имя. К ней можно обращаться через имя указателя, поэтому она называется *ссылочной переменной*.

Типизированные указатели

Указатели, содержащие адрес, по которому записана переменная заранее определенного типа, называются *типизированными*. Для объявления типизированного указателя используется знак $\wedge$, который помещается перед соответствующим типом.

Синтаксическая диаграмма определения типа указателя такова:



В соответствии с диаграммой для объявления типа указателя P в качестве следует записать:

```
type
P = ^integer;
```

Тип *integer* в данном примере является базовым типом. Имея в программе определение типа указателей (ссылочного типа), можно по общим правилам описать переменные этого типа. При этом ссылочные типы в описаниях переменных можно задавать как посредством идентификаторов, так и явно, например:

```
var
  P1, P2 : P; {Тип P введен выше}
  R : ^Byte;
```

Описание типов указателей — единственное исключение из общего правила согласно которому все идентификаторы должны быть описаны перед использованием. Однако если базовый тип является еще не объявленным идентификатором, то он должен быть объявлен в той же самой части объявления, что и тип «указатель».

Например:

```
RecPtr = ^RecordType;
RecordType=record
  Name : string[20];
  Number : integer;
end;
```

В данном примере *RecPtr* описывается как указатель на переменную *RecordType*. Базовый тип *RecordType* описывается в той же самой последовательности определений типов, что и тип *RecPtr*.

Реально значения ссылочных типов содержат адреса расположения в памяти конкретных значений базового типа. В персональном компьютере адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются *сегментом* и *смещением*. Сегмент — это участок памяти, имеющий длину 65536 байт (64 Кбайт) и начинающийся с физического адреса, кратного 16 (0, 16, 32, 48 и т. д.). Смещение указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу. Таким образом, по своей внутренней структуре любой указатель представляет собой совокупность двух слов (данных типа *word*), трактуемых как сегмент и смещение, а абсолютный адрес образуется следующим образом: сегмент*16+смещение. Отсюда видно, что сегменты могут перекрываться, т. е. один и тот же абсолютный адрес может образовываться несколькими способами, например:

Сегмент	\$0020	—1	\$001F
Смещение	\$0010	+16	\$0020
Абсолютный адрес	\$00210		\$00210

Такая многозначность является следствием того, что из 16 + 16 бит сегмента и смещения для образования адреса используются только 20 бит. Поэтому при сравнении адресов, сгенерированных с помощью стандартной функции *Ptr*, нужно быть осторожным. Адреса, генерируемые с помощью стандартных процедур *New* и *GetMem*, всегда нормализуются, т. е. смещение всегда лежит в диапазоне \$0000..\$000F.

Для присвоения переменной ссылочного типа некоторого значения можно воспользоваться унарной операцией взятия указателя, которая состоит из знака этой операции — символа $@$ (амперсанд) и одного операнда-переменной. Например, если имеется описание переменной *I* целого типа:

```
var
  I : integer;
```

то применение этой операции к переменной *I*: $@I$ дает в качестве результата значение типа «указатель на целое». Аналогичный результат получится и в результате использования операции $P = ^integer$.

ПРИМЕЧАНИЕ

Операция взятия указателя допустима для любых переменных, в том числе для элементов массивов, полей записи и т. д. Например, если есть описание `var A : array[1..10] of integer`; то конструкция $@A[I]$ означает «указатель на I-е целое в массиве A» и также может участвовать в присваивании: $P := @A[I]$.

Ссылочные типы можно образовывать от любых типов, поэтому допустимо определение вида «указатель на указатель».

Среди всех возможных указателей в Turbo Pascal выделяется один специальный указатель, который «никуда не указывает». Можно представить такую ситуацию: в адресном пространстве оперативной памяти компьютера выделяется один адрес, в котором заведомо не может быть размещена никакая переменная. На это место в памяти и ссылается такой *нулевой* (или *пустой*) указатель, который обозначается словом `Nil`. Указатель `Nil` считается константой, совместимой с любым ссылочным типом, поэтому его значение можно присваивать любому указателю. Обычно значение `Nil` присваивают указателю, когда требуется отменить его значение или в начале программы. Это позволяет проверять значение указателя, прежде чем присваивать ему какое-либо значение.

Нетипизированный указатель (pointer)

В Turbo Pascal можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип `pointer`. Он обозначает нетипизированный указатель, т. е. указатель, который не указывает ни на какой определенный тип. С помощью нетипизированных указателей удобно динамически размещать данные, структура и тип которых меняются в ходе программы.

ВНИМАНИЕ

Переменные типа `pointer` не могут быть разыменованы; указание символа `^` после такой переменной вызывает ошибку. Как и значение, обозначаемое словом `Nil`, значения типа `pointer` совместимы со всеми другими типами указателей.

Значения ссылочных типов допускают две операции сравнения: на равенство и неравенство, например `@X <> @Y` или `P1 = P2`. Два указателя равны только в том случае, если они ссылаются на один и тот же объект.

ПРИМЕЧАНИЕ

При сравнении указателей в Turbo Pascal просто сравниваются сегменты и смещения. В соответствии со схемой размещения сегментов процессоров 80x86 два логически различных указателя могут фактически указывать на одну и ту же физическую ячейку памяти.

Доступ к переменной по указателю

Для доступа к переменной имеются две возможности. Первая состоит в использовании идентификатора переменной. Например, чтобы увеличить значение переменной `I`, на которую указывает указатель `P`, можно записать `I:=I+2`. Для реализации второго, косвенного доступа к переменной по указателю, производится разыменование указателя. Правило разыменования можно сформулировать следующим образом: для того чтобы по указателю

переменную получить доступ к самой переменной, нужно после переменной-указателя поставить знак `^`. Так, запись `Writeln(Int2^);` означает «напечатать значение переменной, на которую ссылается указатель `Int2`».

Поэтому чтобы увеличить значение переменной `I`, на которую указывает указатель `P`, используя косвенный доступ к переменной по указателю, можно записать: `P^ := P^ + 2`.

ПРИМЕЧАНИЯ

Разыменование допускается для любых ссылочных типов. В случае использования «указателя на указатель» возможно многократное разыменование.

Разыменование считается некорректным, если ссылочная переменная имеет значение `Nil`, т. е. в этом случае не существует переменной, на которую ссылается указатель.

Управление динамической памятью

Вся динамическая память в Turbo Pascal рассматривается как сплошной массив байтов. Физически динамическая память располагается в старших адресах непосредственно за областью памяти, занимаемой телом программы.

Указатель на начало динамической памяти хранится в стандартной переменной `HeapOrg`, указатель на конец — в переменной `HeapEnd`. На текущую границу незанятой динамической памяти указывает `HeapPtr`.

Для управления динамической памятью используются следующие стандартные процедуры и функции.

Процедуры динамического распределения памяти

Процедура	Описание
<code>Dispose</code>	Уничтожает динамическую переменную
<code>FreeMem</code>	Уничтожает динамическую переменную заданного размера
<code>GetMem</code>	Создает новую динамическую переменную заданного размера и устанавливает переменную-указатель для нее
<code>Mark</code>	Записывает в переменную-указатель состояние динамической памяти
<code>New</code>	Создает новую динамическую переменную и переменную-указатель для нее
<code>Release</code>	Возвращает динамическую память в заданное состояние

Функции динамического распределения памяти

Функция	Описание
<code>MaxAvail</code>	Возвращает размер наибольшего непрерывного свободного блока динамической памяти, соответствующей размеру наибольшей динамической переменной, которая может быть распределена в момент вызова <code>MaxAvail</code>
<code>MemAvail</code>	Возвращает количество имеющихся в динамической памяти свободных байтов

Функции для работы с указателями и адресами

Функция	Описание
Addr	Возвращает адрес заданного объекта
CSeg	Возвращает текущее значение регистра CS
DSEg	Возвращает текущее значение регистра DS
Ofs	Возвращает значение смещения для заданного объекта
Ptr	Преобразует базовый адрес сегмента и смещение в значение типа «указатель»
Seg	Возвращает адрес сегмента для заданного объекта
SPtr	Возвращает текущее значение регистра SP
SSeg	Возвращает текущее значение регистра SS

Основные действия над динамическими переменными — создание и уничтожение — реализуются в Turbo Pascal при помощи стандартных процедур New и Dispose.

Процедура New предназначена для создания динамических переменных определенного типа. Она отводит новую область в динамической памяти для данной динамической переменной и сохраняет адрес этой области в переменной-указателе. Переменной-указателю можно присвоить значение и с помощью оператора @ или функции Ptr. Оператор @ устанавливает переменную-указатель на область памяти, содержащую существующую переменную, включая и те переменные, которые имеют идентификаторы. Функция Ptr устанавливает переменную-указатель на определенный адрес в памяти. Например:

```
New(Int1); P1 := @X; Ptr($40, $49);
```

Для освобождения области динамической памяти предназначена процедура Dispose, принимающая в качестве параметра указатель на динамическую переменную, причем эта переменная должна быть ранее размещена в динамической памяти посредством New, а тип размещенной переменной должен совпадать с базовым типом параметра процедуры Dispose.

Например, Dispose(Int1); освобождает выделенный в предыдущем примере фрагмент динамической памяти таким образом, что его можно снова использовать, если потребуется.

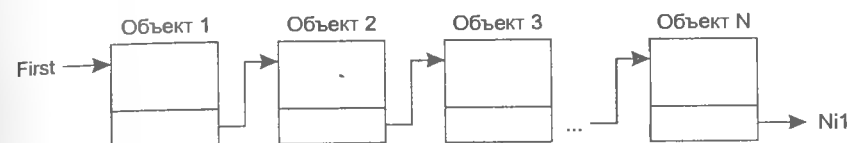
Если в программе не предполагается использование динамической памяти, то в Turbo Pascal рекомендуется уменьшить объем доступной памяти для создаваемой программы с помощью меню Options/Heap.

ВНИМАНИЕ

При освобождении динамической памяти нужно соблюдать осторожность и использовать New/Dispose, либо New/Mark/Release, либо GetMem/FreeMem, но ни в коем случае не смешивать эти процедуры.

Использование указателей для организации связанных списков

Чаше всего указатели используются для ссылки на записи, тем самым достигается значительная экономия памяти. Если же сама запись содержит в себе поле-указатель, указывающий на следующую за ней запись, то это позволяет образовать *связные списки* — структуры, в которых отдельные записи последовательно связаны друг с другом. В приведенном ниже примере используются записи, в которых наряду с данными об автомобиле имеется указатель на следующую запись, благодаря чему получается связный список.



Одно из полей каждого объекта связного списка имеет тип «указатель» и указывает на очередной объект в списке. Указатель на первый объект содержится в переменной First, а последний объект содержит указатель на Nil.

Упражнения

Упражнение 1. Изучите текст программы, которая демонстрирует использование указателей массива.

```

program d_point1; {Пример указателя массива и указателя компонента}
type
  feld = array[1..10] of integer; {Описание типа массива из
                                   10 целых чисел}
var
  I   integer; {Целая переменная для указания индекса элемента массива}
  F   feld;    {Переменная типа feld (массив из 10 целых чисел)}
  Zf  ^feld;   {Указатель на массив}
begin
  for I:=1 to 10 do F[I]:=10*I; {Генерировать массив}
  Zf:=@F;                      {Присвоить указателю Zf адрес начала
                               размещения массива в памяти}

  Writeln(Longint(@F));
  Writeln(Zf^[1]);             {Напечатать значение 1-го элемента массива.
                               на который указывает Zf}
  Zf:=@F[2];                  {Присвоить указателю Zf адрес размещения
                               2-го элемента массива}
  Writeln(Zf^[1]);             {Напечатать значение 1-го элемента массива.
                               на который указывает Zf}
  Zf:=Ptr(Seg(F[3]).Ofs(F[3])+SizeOf(integer));
  
```

```

        {Присвоить указателю Zf адрес
        4-го элемента массива}
    WriteLn(Zf^[1]).    {напечатать значение 1-го элемента массива,
                        на который указывает Zf}
end.

```

Запустите интегрированную среду программирования, введите текст программы d_point1, запишите его на диск и откомпилируйте. Выполните программу в пошаговом режиме и проследите за изменением значений переменных: I, F, Zf, Zf[^]. Обратите внимание на разный результат вывода значений элементов массива на экран при помощи оператора WriteLn(Zf^[1]) из-за изменения значения указателя на массив.

Упражнение 2. Изучите текст программы, которая демонстрирует использование динамической памяти.

```

program D_point2;
type IntPtr = ^integer; {Описание типа указателя в качестве целого IntPtr}
   RecPtr = ^RecordType; {Описание типа указателя в качестве целого RecordType}
   RecordType = record {Описание типа записи RecordType}
       Name : string[20];
       Number : integer;
   end;

var
    Int1, Int2, P1, P2 : IntPtr; {Описание переменных типа IntPtr}
    He : RecPtr; {Описание переменной типа RecPtr}
    X, Y, I : integer; {Переменные целого типа}
    P : ^Byte; {Указатель на данные типа byte}

begin
    P := Ptr($40, $49) {Присвоить P значение ячейки памяти по адресу:
                        сегмент $40, смещение $49 — по этому адресу
                        записан текущий видеорежим}
    WriteLn('Текущий видеорежим :'. P^);
    WriteLn('MemAvail.' байт всего свободно в памяти (размер динамической памяти)');
    WriteLn('Объем наибольшего непрерывного свободного блока динамической памяти
    ='. MaxAvail. ' байт');
    Write('Введите два целых числа :');
    ReadLn(X, Y);
    New(Int1); {Создать новую динамическую переменную Int1 и установить на нее
    указатель}
    Int1^:=X; {Записать по адресу, указанному Int1, значение переменной X}
    WriteLn('X=' X. ' Int1^=' Int1^);
    New(Int2); {Создать новую динамическую переменную Int2 и установить на нее
    указатель}
    Int2^:=Y; {Записать по адресу, указанному Int2, значение переменной Y}
    WriteLn('Y=' Y. ' Int2^=' Int2^);
    {Подсчитать размер динамической памяти для размещения одной
    RecordType и вывести сообщение}
    WriteLn('Для одной записи нужно '. SizeOf(RecordType). ' байт');

```

```

New(He):    {Создать новую динамическую переменную He и установить
            на нее указатель}
Write('Введите Ваши фамилию и имя :');
ReadLn(He.Name):    {Считать в динамическую переменную
                    значение фамилии и имени}

Write('Сколько Вам лет :');
ReadLn(He.Number):    {Считать в динамическую переменную
                    значение возраста}
                    {Напечатать на экране значения полей
                    динамической переменной He типа RecPtr}
WriteLn('He.Number=' He.Number. ' He.Name=' He.Name);
                    {Выполнить сравнение значений динамических
                    переменных, на которые указывают He.Number
                    и Int1^ и выполнить соответствующие
                    операции}
if He.Number>Int1^ then Y:= He.Number
else Y:=Int1^;

WriteLn('Y=' Y);
Dispose(Int1);    {Освободить память, отведенную ранее под
                  динамическую переменную Int1}

Dispose(Int2);
Dispose(He);
P1 := @X;    {Теперь присвоить указателю P1 адрес переменной X}
P2 := Addr(Y); {Присвоить указателю P2 адрес размещения в
                динамической памяти переменной Y}
WriteLn(P1^): {Напечатать значение переменной X}
P2 := P1;    {X --> P2}
Y := P2^;    {То же, что и Y := X;}
                {Сравнить значения адресов, по которым в динамической
                памяти записаны переменные X и Y}
if @X <> @Y then WriteLn('Значения адресов переменных X и Y не равны');
WriteLn(Y);    {Выводит значение Y, равное значению X}
end.

```

Запустите интегрированную среду программирования, введите текст программы d_point2, запишите его на диск и откомпилируйте. Выполните программу в пошаговом режиме и проследите за изменением значений переменных: MemAvail, MaxAvail, HeapOrg, HeapEnd, HeapPtr, X, Y, He[^]. Обратите внимание на изменение значений переменных MemAvail, MaxAvail при создании и уничтожении динамических переменных.

Упражнение 3. Изучите текст программы, которая считывает файл целых чисел, состоящий из нескольких последовательностей чисел, каждая из которых оканчивается числом -1, и затем выводит эти последовательности чисел в выходной файл, но внутри каждой из них расставляет числа в обратном порядке:

```

program d_point3;
type
    Pointer = ^ DataSet; {Определение указателя на запись DataSet}

```

```

DataSet =record      {Определение структуры записи}
    Data : integer;   {Поле – целое число}
    Point : Pointer;  {Поле – нетипизированный указатель}
end;

var
    R1, R2 : pointer;   {Нетипизированные указатели}
    I, K : integer;     {Вспомогательные переменные}
    Inp, Out: file of integer; {Файловые переменные Inp – исходный файл целых чисел, Out – выходной файл целых чисел}
begin
    {Фрагмент создания исходного файла – при первом запуске отключите комментарии}
    { Assign(Inp, 'Input.dat');
    Rewrite(Inp);
    for K:=1 to 20 do {Создать файл из 20 чисел}
    begin
        Write('Введите '.K, '-е целое число :');
        Readln(I);
        Write(Inp, I); {Записать в исходный файл очередное число}
    end;
    Close(Inp); {Завершение создания исходного файла чисел}
    Assign(Inp, 'Input.dat ');
    Reset(Inp); {Открыть исходный файл Input.dat}
    Assign(Out, 'Output.dat ');
    Rewrite(Out); {Открыть исходный файл Output.dat}
    while not Eof(Inp) do {Пока не будет обнаружен конец исходного файла}
    begin
        R1:=Nil; {Установить указатель R1 на Nil}
        Read(Inp, I); {Читать из исходного файла число в переменную I}
        while I<>-1 do {Пока значение переменной I не равно -1 (конец
последовательности)}
        begin
            New(R2); {Создает новую динамическую переменную и устанавливает на нее
переменную-указатель}
            R2^.Data:=I; {Записываем в поле Data динамической переменной DataSet .
адрес которой указывает R2 (первой в создаваемой в динамической памяти цепочки).
значение переменной I – очередного числа последовательности}
            R2^.Point:=R1; {Записываем в поле Point динамической переменной DataSet
адрес которой указывает R2. значение указателя}
            R1:=R2; {Теперь указатель R1 указывает на новый объект в цепочке чисел}
            Read(Inp, I); {Читать из исходного файла число в переменную I}
        end;
        R2:=R1; {Указать начало следующей последовательности чисел}
        while R2<>Nil do {Пока значение указателя R2 <> Nil}
        begin
            Write(Out, R2^.Data); {Записать в выходной файл значение поля Data
динамической переменной, на которую указывает R2}
            Write(R2^.Data); {Напечатать на экране (для контроля) значение поля Data
динамической переменной, на которую указывает R2}

```

```

R2:= R2^.Point {Присваиваем R2 значение поля Point динамической
переменной DataSet, указывающей на следующее число в последовательности}
end;
Write(Out, I); {Записать в выходной файл I=-1}
Writeln;
end;
Dispose(R1); {Освободить динамическую память}
Close(Inp); {Закрыть исходный файл}
Close(Out); {Закрыть выходной файл}
end.

```

Запустите интегрированную среду программирования, введите текст программы d_point3, запишите его на диск и откомпилируйте. Выполните программу в пошаговом режиме. При первом запуске программы исключите фигурные скобки, обозначающие комментарии в фрагменте создания исходного файла, и введите 20 целых чисел, разделяя их на последовательности числом -1, например: 1, 3, 5, -1, 0, 3, 5, 6, -1, 9, 2, 4, -1, 2, 33, 4, 21, 33, 7, -1. Перед повторным запуском программы отключите фрагмент создания исходного файла, чтобы не вводить заново исходную последовательность чисел. Запустив программу в пошаговом режиме, проследите за изменением значений переменных MaxAvail, R1, R2^.Point, R2, I, R2^.Data.

Упражнение 4. Изучите текст программы, в которой создается связный список из записей, содержащих сведения об автомобилях, и иллюстрируются некоторые операции со связными списками: запись первого объекта в список; удаление первого объекта из списка; просмотр всего списка; удаление объекта, следующего за указанным.

```

program d_point5; {Пример использования указателей для обработки связного списка}
uses Crt;
type
    NameStr = string[20];
    Link = ^Auto;
    Auto = record
        Name : NameStr; {Марка автомобиля}
        Speed : real; {Скорость}
        Next : Link; {Поле для связи со следующим объектом в списке}
    end;
var
    P, First : Link; {Указатели на запись: текущую, первую}
    NamFind : NameStr; {Марка автомобиля для поиска}
    V : 0..4; {Селектор меню}
    EndMenu : boolean; {Окончание вывода меню}
function FindName(FN:NameStr) : Link;
{Поиск объекта с именем FN, по результатам
поиска возвращает указатель на найденный
объект или Nil, если объект не найден}
var
    Curr : Link;

```

```

begin
  Curr:=First;      {Установить указатель на первом объекте в списке}
  while Curr <> Nil do {Повторять, пока не дойдем до конца списка}
    if Curr^.Name = FN then {Если нашли заданный объект}
      begin
        FindName:=Curr; {Возвращаем значение указателя на него}
        Exit;           {Завершаем функцию}
      end
    else
      Curr:=Curr^.Next; {Перейти к следующей записи}
      FindName:=Nil;    {В списке нет искомого объекта}
    end;
  end;
  procedure AddFirst(A:Link); {Добавление записи первой в связный список}
  begin
    A^.Next:=First; {Новый объект первый в списке}
    First:=A;       {Голова списка ссылается на новый объект}
  end;
  procedure DelFirst(var A:Link); {Удаление первого объекта из списка}
  begin
    A:=First;
    First:=First^.Next; {Теперь First указывает на тот объект, на который ранее
    ссылался объект A}
  end;
  procedure DelAfter(Old:Link; var A:Link); {Удаление из списка объекта, стоящего
  после объекта Old }
  begin
    A:=Old^.Next; {Переменной A присваивается значение указателя на удаляемый
    объект}
    Old^.Next:=Old^.Next^.Next; {Теперь Old указывает на тот объект, на который
    ранее ссылался следующий за ним объект, а объект A исключен из связного списка}
  end;
  procedure InpAvto; {Ввести данные об объекте}
  begin
    P:=New(Link); {Создать очередной объект типа Auto}
    Write('Введите марку автомобиля :');
    Readln(P^.Name);
    Write('Максимальная скорость :');
    Readln(P^.Speed);
    AddFirst(P); {Вызов процедуры добавления записи, на которую ссылается
    указатель P (P – фактический параметр, A – формальный параметр-значение)}
  end;
  procedure MyList; {Вывести на экран все объекты из связного списка}
  var
    Curr Link; {Локальная переменная – указатель на очередной объект}
  begin
    Curr:=First; {Установить указатель на первом объекте в списке}
    while Curr <> Nil do {Повторять, пока не дойдем до конца списка}

```

```

begin
  WriteLn('Марка : '.Curr^.Name.' скорость : '. Curr^.Speed);
  Curr:=Curr^.Next; {Перейти к очередному объекту связного списка}
end;
write('Вывод списка окончен. Нажмите Enter');
Readln.
{Конец MyList}
{Основная программа}
begin
  Lagin
  New(P); {Создать новую динамическую переменную и установить на нее
  переменную-указатель}
  EndMenu:=False.
  repeat {Очищать экран и выводить меню до тех пор, пока EndMenu<>True}
    ClrScr.
    WriteLn('Укажите вид работы:');
    WriteLn('1. Запись первым в список');
    WriteLn('2. Удаление первого объекта из списка');
    WriteLn('3. Просмотр всего списка');
    WriteLn('4. Удаление объекта, следующего в списке за указанным');
    WriteLn('0. Окончание работы');
  Readln(V);
  case V of {Вызов разных процедур в зависимости от выбора пункта меню}
    1 : InpAvto; {Ввод данных об объекте}
    2 : DelFirst(P); {Удаление первого в списке}
    3 : MyList; {Вывод списка всех элементов связного списка}
    4 : begin {Удаление объекта, следующего за указанным}
        Write('Введите марку автомобиля, за которым следует удаляемый
        из списка ');
        Readln(NamFind);
        DelAfter(FindName(NamFind).P); {Вызов процедуры DelAfter
        с параметрами: функцией FindName(NamFind) и указателем P}
      end;
    else
      EndMenu:=True; {Завершить вывод меню}
  end;
  until EndMenu; {Если EndMenu=True, то завершить вывод меню на экран}
  Dispose(P). {Уничтожить динамическую переменную P и освободить память
  в динамической памяти}
end.

```

Запустите интегрированную среду программирования, введите текст программы d_point5, запишите его на диск и откомпилируйте. Выполните программу в пошаговом режиме с трассировкой процедур и проследите за изменением значений переменных MaxAvail, Curr, P, First^, Old^, A, A^, A^.Next. Каждую операцию по изменению связного списка (запись первого объекта в список; удаление первого объекта из списка; удаление объекта, следующего за указанным) сопровождайте выполнением просмотра всего списка для отслеживания изменений в списке.

Как можно заметить, в программе моделируется поведение, напоминающее стек: конец списка является основанием стека, а новые элементы добавляются в вершину стека.

Контрольные вопросы и задания

Вопросы

1. Какие переменные называются статическими, в каких случаях они применяются, почему их необходимо предварительно описывать?
2. Какие переменные называются динамическими, в каких случаях они используются?
3. Что такое указатель и базовый тип? Чем отличаются типизированные и нетипизированные указатели?
4. Каковы особенности описания типов указателей?
5. Каково назначение указателя Nil?
6. Объясните синтаксическую диаграмму определения типа указателя.
7. Для чего применяется и как задается операция взятия указателя?
8. Что такое разыменование указателя? В чем заключается правило разыменования?
9. Что называется динамической памятью? В какой области памяти компьютера она располагается? Какие значения имеют стандартные переменные HeapOrg, HeapEnd, HeapPtr?
10. Какие стандартные процедуры и функции используются для управления динамической памятью?
11. Каковы особенности применения сочетания стандартных процедур при освобождении динамической памяти?
12. Даны следующие описания:

```
type ref = ^integer;
var P, Q : ref;
```

Переменные P и Q имеют значения, как показано на рисунке.



Ответьте на вопросы:

- 1) что является значением переменной P: ссылка на объект (переменную) целого типа или сам этот объект? Что обозначает переменная P^: ссылку на объект целого типа, сам этот объект или целое число 5? Каковы типы переменных P и P^?

- 2) что будет выведено на печать в результате выполнения следующих операторов?

```
P^ := Q^;
if P=Q then P:=Nil
    else if P^=Q^ then Q:=P;
if P=Q then Q^:=4;
WriteLn(P^);
```

Задания

1. Имеется программа:

```
program zadanie1;
var X : ^boolean; Y : boolean;
begin
  {a} New(X);
  {b} X^:=True; Y:=not X^;
  {c} Dispose(X);
  {d} WriteLn(Y);
end.
```

Ответьте на следующие вопросы:

- 1) какие переменные существуют в каждой из точек a, b, c, d и каковы их значения в эти моменты?
 - 2) можно ли переменной X присвоить ссылку на переменную Y? Можно ли с помощью процедуры Dispose уничтожить переменные X и Y?
2. Найдите ошибки в следующей программе:

```
program Errors;
var A, B : ^integer;
begin
  if A=Nil then Read(A);
  A^:=5;
  B:=Nil;
  B^:=2;
  New(B);
  Read(B^);
  WriteLn(B.B^);
  New(A);
  B:=A;
  Dispose(A);
  B^:=4;
end.
```

3. Дан список случайных целых чисел. Переверните список, т. е. расставьте все числа в обратном порядке. Подсчитайте среднее арифметическое элементов списка. Создайте два новых списка, в один из которых запишите все элементы большие пяти, в другой — остальные элементы исходного списка.

4. Дан список слов. Выведите на экран слова списка, которые:
 - 1) оканчиваются и начинаются с одной и той же буквы;
 - 2) начинаются с той же буквы, что и следующее слово;
 - 3) совпадают с последним или первым словом.
5. Дан текстовый файл. Распечатайте текст в обратном порядке слов.
6. Дан файл сведений об учащихся (фамилия, имя, возраст), расположенных по алфавиту фамилий. Создайте новый файл, в котором эти сведения будут записаны в обратном порядке.
7. Дан текстовый файл. Замените все вхождения указанного слова на заданное слово.
8. Дан файл картотеки сведений об автомобилях (марка, номер и фамилия владельца). Произведите с ним следующие действия:
 - 1) найдите фамилии владельцев, номера автомобилей указанной марки и их количество;
 - 2) найдите фамилию владельца по указанному номеру автомобиля;
 - 3) исключите из картотеки сведения об автомобиле заданного владельца.

часть II

Введение в программирование в Delphi

Глава 11

Введение в объектно-ориентированное программирование

Основные понятия объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) зародилось в языках программирования Pascal, Ада, Smalltalk, C++. До появления ООП технология создания компьютерных программ базировалась на *процедурном программировании*, в котором основой программ являлись функции и процедуры, т. е. действия. Как вы имели возможность убедиться в первой части этой книги, программист определял, какие действия и вычисления нужны для решения поставленной задачи, затем описывал эти действия в виде процедур и функций и объединял их в программу. Созданная таким образом компьютерная программа отличалась четким алгоритмом работы — последовательностью действий по достижению поставленной цели.

Объектно-ориентированное программирование представляет собой отличный от процедурного способ программирования, который напоминает процесс человеческого мышления. В объектно-ориентированном программировании главной отправной точкой при проектировании программы является не процедура, не действие, а *объект*. Такой подход представляется достаточно естественным, поскольку в реальном мире мы имеем дело именно с объектами, взаимодействующими друг с другом. Объектно-ориентированное программирование базируется на трех основных принципах: наследование, инкапсуляция и полиморфизм. Программа, построенная по этим принципам, — это не последовательность операторов, не некий жесткий алгоритм, а совокупность объектов и способов их взаимодействия. Обмен информацией между объектами происходит посредством *сообщений*. С точки зрения ООП взаимодействие пользователя с программой — это тоже взаимодействие двух объектов — программы и человека, которые обмениваются друг с другом определенными сообщениями.

Что такое объекты?

Весь окружающий нас мир состоит из объектов, предметов живой и неживой природы, которые представляются как единое целое, а отдельные части объектов вступают в сложное взаимодействие друг с другом. При структурном подходе программист обычно разделяет описываемый объект на составные части (структурирует его), стараясь описать свойства отдельных частей, не вдаваясь в подробности их взаимодействия, что, вообще говоря, не является лучшим способом программирования.

Назовем *объектом* понятие, абстракцию или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Например, объектами являются: форточка, яблоко. Петр Сидоров, дело № 7461, банковский счет и т. д. Все объекты могут быть идентифицированы: пусть у нас есть два яблока, имеющие одинаковый цвет, форму, вес и вкус; все равно это — два яблока, а не одно, в чем легко убедиться, съев одно из них (другое останется). Между объектами можно установить отношение тождества: объекты, удовлетворяющие этому отношению, одинаковы (тождественны), как вышеупомянутые яблоки.

Объекты представляют высший уровень абстракции данных. Объект можно разделить на части, но тогда он перестанет быть объектом. Отношения частей к целому и взаимоотношения между отдельными частями становятся понятнее тогда, когда части объединяются в единое целое. Это называется *инкапсуляцией* и является очень важным понятием.

Не менее важным является и тот факт, что объекты могут наследовать характеристики и поведение других объектов, называемых *порождающими*, или *родительскими*, объектами (или «предками»). С введением понятия наследования происходит качественный скачок, поскольку наличие механизма наследования является самым существенным различием между обычным программированием на Pascal и объектно-ориентированным программированием в Delphi.

Классы объектов

В Object Pascal наряду с понятием объекта широко используется понятие класса. *Классом* называют особую структуру, которая может иметь в своем составе поля, методы и свойства. Класс выступает в качестве объектного типа данных, а объект — это конкретный существующий в памяти компьютера экземпляр класса. Например, два яблока из предыдущего примера принадлежат одному и тому же классу объектов (именно с этим связана их одинаковость). Цвет, форма, вес и вкус яблока — это его атрибуты: совокупность атрибутов и их значений (например, красное, овальное, стограммовое, кисло-сладкое) характеризует объект. Объединение объектов в классы определяется семантикой (смыслом). Так, например, объекты «конюшня» и «лошадь» могут иметь одинаковые атрибуты: цену и возраст. При этом они могут относиться к од-

ному классу, если рассматриваются в задаче просто как товар, либо к разным классам, что более естественно. Объединение объектов в классы позволяет ввести в задачу абстракцию и рассмотреть ее в более общей постановке. Класс имеет имя (например, «лошадь»), которое относится ко всем объектам этого класса. Кроме того, в классе вводятся имена атрибутов, которые определены для объектов. В этом смысле описание класса аналогично описанию типа структуры (записи); при этом каждый объект имеет тот же смысл, что и экземпляр структуры (переменная или константа соответствующего типа).

Классы имеют поля (как тип данных `Record`), свойства (напоминающие поля, но имеющие дополнительные описатели, определяющие механизмы записи и считывания данных, что позволяет повысить строгость декларирования внутренней структуры класса) и методы (подпрограммы, которые обрабатывают поля и свойства класса).

Базовым классом для всех объектов в Delphi является класс `TObject`. Он инкапсулирует основные функции, свойственные всем объектам Delphi. Все классы в Delphi являются прямыми или косвенными наследниками `TObject`. Прямое наследование используется только при объявлении простых классов, объекты которых не являются компонентами, не могут присваиваться друг другу и не участвуют в операциях обмена с потоками. Подавляющее большинство составляют классы, являющиеся косвенными наследниками `TObject` и прямыми наследниками промежуточных классов.

ПРИМЕЧАНИЕ

Если при объявлении нового типа объектов не указывается класс-предок, то предком нового класса считается класс `TObject`.

Иерархия объектов класса

Обычно, классифицируя некоторый объект, мы задаем следующие вопросы: чем этот объект похож на другие объекты общего класса и чем он отличается от других объектов? Каждый конкретный класс имеет свои особенности поведения и характеристик, определяющих этот класс. Например, класс геометрических фигур можно разделить на два подкласса: плоские фигуры и объемные фигуры. Плоские фигуры могут иметь вершины и не иметь их. Плоскими фигурами, не имеющими вершин, являются окружности и эллипсы.

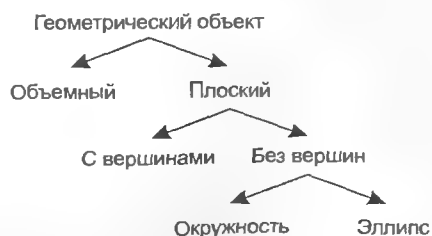


Рис. 11.1. Фрагмент графического представления иерархии объектов класса

Задавая себе приведенные выше вопросы, программист продвигается от вершины иерархического дерева класса и проходит по дочерним подклассам. Наивысший уровень — самый общий, а вопросы здесь самые простые, например, являются ли фигуры плоскими или объемными. Каждый последующий уровень является более специфическим, чем предыдущий, и менее общим. На самом последнем уровне программист определит цвет, стиль заполнения, величину радиуса окружности и другие конкретные детали фигур.

При использовании ООП следует помнить, что если характеристика однажды определена, то все категории, расположенные ниже данного определения, тоже будут содержать эту характеристику. Поэтому если определена окружность, то нет необходимости узнавать, сколько у нее вершин, так как она относится к подклассу фигур, не имеющих вершин. Объектно-ориентированное программирование является наилучшим инструментом для построения иерархических деревьев структур данных. Одной из важных особенностей, которые объектно-ориентированное программирование добавляет к традиционным языкам типа Pascal, является механизм, с помощью которого типы данных могут наследовать характеристики более простых, т. е. более общих типов. Этим механизмом является наследование.

Наследование

В терминах Pascal объект наиболее схож с типом `Record`, который является структурированным типом для объединения нескольких связанных элементов под одним именем.

ПРИМЕЧАНИЕ

Примеры использования типа `Record` рассмотрены в главе 8 первой части данной книги.

Предположим, что требуется написать программу, подсчитывающую размер стипендий и заработной платы в институте или университете. Переменная `TPerson`, содержащая данные об именах студентов или сотрудников, дате и размере выплаты, могла бы выглядеть следующим образом:

```

TPerson = Record
    Name : String[30];
    Date : String[10];
    Rate : Real;
end;
  
```

Каждое значение, присвоенное переменной `TPerson`, является экземпляром типа `record`.

ПРИМЕЧАНИЕ

В дальнейшем термин «экземпляр» будет часто использоваться при описании программ, созданных объектно-ориентированным методом.

TPerson представляет два уровня абстракции. Можно рассматривать поля Name, Date и Rate по отдельности, а когда речь идет о полях, работающих одновременно для описания конкретного человека, можно рассматривать их совокупность как TPerson.

Предположим, что программа должна учитывать выплаты денег студентам, преподавателям и сотрудникам кафедры. В каждой группе выплаты производятся особым способом. Можно создать другой тип записи для каждой группы. Например, для получения данных о том, сколько денег должен получить студент, необходимо в первую очередь знать его средний балл. Можно построить запись TStudent вида:

```
TStudent = Record
    Name : String[30];
    Date : String[10];
    Rate : Real;
    Ball : Real;
```

end;

Однако можно сохранить тип TPerson путем создания поля Student типа TPerson внутри типа TStudent.

```
TStudent = Record
    Student : TPerson;
    Ball : Real;
```

end;

Такая конструкция удобна и проста, поэтому постоянно используется в программировании. Но она не учитывает специфику данных, обрабатываемых в программе. Необходимо учесть, что выплата денег студентам отличается от выплат другим лицам. Естественно, что студент, как и преподаватель или другой служащий кафедры, имеет имя, фамилию, год рождения и ему полагается определенная сумма денег.

Для студента тип TStudent должен содержать все поля, которые имеются в записи TPerson, при этом тип TStudent является типом-потомком для типа TPerson. TStudent наследует все, что принадлежит TPerson, и, кроме того, содержит новые поля, которые делают TStudent уникальным.

Процесс, с помощью которого один тип наследует характеристики другого типа, называется *наследованием*. В Object Pascal все классы являются потомками класса TObject. В данном примере наследующий тип (TStudent) называется *порожденным типом* или *потомком*, а тип, который наследуется (TPerson), называется *родительским типом*, или *предком*.

Итак, мы познакомились с новой категорией структуры данных, связанной с записями и имеющей механизм наследования. Типы данных в этой новой категории определяются с помощью зарезервированного слова *object*. Объектный тип может быть определен как полный, самостоятельный тип, подобно описанию записей в Pascal, но может определяться и как потомок существ-

ующего типа объекта путем указания после зарезервированного слова *object* заключенного в скобки имени родительского типа.

В приводимом примере два связанных типа объектов могли бы определяться следующим образом:

```
type
TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
```

end;

Здесь TPerson является родительским типом, а TStudent — дочерним. Фактически, процесс наследования может продолжаться сколь угодно долго. Для дочернего типа TStudent при необходимости можно определить еще один дочерний тип. Все типы, наследующие тип TPerson, называются его *дочерними типами*, но TStudent является *непосредственным дочерним типом* для TPerson, а TPerson является *непосредственным родителем* типа TStudent.

Основное различие между переменными типа *object* и *record* состоит в том, что использование зарезервированного слова *object* при создании объектов в Pascal позволяет не указывать явно поля Name, Date и Rate типа TPerson в типе TStudent, при этом тип TStudent будет содержать эти поля благодаря свойству наследования.

Поля объектов

К полю объекта можно обратиться, как и к полю обычной записи, с помощью оператора *with*, либо используя префикс с именем объекта, например:

```
Student.Ball := 4.5;
with Student do begin
    Name := 'Иванов Николай Петрович';
    Date := '25-06-1995';
end;
```

Даже если поля Name, Date и Rate не являются частью описания типа TStudent, благодаря тому, что они наследуются от типа TPerson, на них можно ссылаться как на описанные в TStudent:

```
Student.Name := 'Николай Иванов';
```

Как правило, объект — это сложная конструкция, поведение и свойства составных частей которой находятся во взаимодействии. К полям объекта можно обратиться непосредственно, но лучше этого избегать. Принципы объектно-ориентированного программирования требуют, чтобы поля объектов были исключены из исходного кода, насколько это возможно. Поля метода можно объявить как скрытые, ограничив возможность доступа к ним пределами модуля, в котором они определены. Для обеспечения надежности функционирования программы применяется правило *инкапсуляции*: исклю-

чение прямого доступа к полям объекта при помощи непосредственных операций чтения и обновления их содержимого; доступ к полям объекта осуществляется посредством вызова соответствующих методов.

Наиболее рациональным способом получения доступа к полям данных в Pascal является использование процедур или функций, описанных внутри объекта

Операции и методы

Функция (или преобразование), которую можно применять к объектам данного класса, называется *операцией*. Примеры операций: проверить, снять, поместить (для объектов класса «счет»), открыть для чтения, читать, закрыть (для объектов класса «файл») и т. п.

ПРИМЕЧАНИЕ

Все объекты одного класса используют один и тот же экземпляр каждой операции (т. е. увеличение количества объектов некоторого класса не приводит к увеличению количества загруженного программного кода). Объект, из которого вызвана операция, передается ей в качестве неявного аргумента (параметра).

Одна и та же операция может, вообще говоря, применяться к объектам разных классов. Такая операция называется *полиморфной*, так как она может иметь разные формы для разных классов. Например, для объектов классов «вектор» и «комплексное число» можно определить операцию +; эта операция будет полиморфной, так как для сложения векторов и сложения комплексных чисел выполняются разные действия.

Методы. Инициализация полей объектов

Обычно при работе с записями возникает проблема инициализации полей записи. Предположим, имеется следующая структура:

```
TPerson = Object
  Name : String[30];
  Date : String[10];
  Rate : Real;
end;
```

Для присвоения полям Name, Date и Rate начальных значений можно использовать оператор with, но при необходимости инициализировать более одной записи типа TPerson придется использовать большое число операторов with, которые будут выполнять одни и те же действия. Поэтому естественным является создание инициализирующей процедуры, которая обобщает применение оператора with к любому экземпляру типа TPerson, передаваемого в качестве параметра:

```
procedure Init(var Person:TPerson; Nm, Dt: String; Rt: Real);
begin
  with Person do begin
```

```
Name := Nm;
Date := Dt;
Rate := Rt;
```

```
end;
```

```
end;
```

Процедура Init, включенная в объект специально для обслуживания типа TPerson, называется методом, т. е. *метод* — это процедура или функция, включенная в объект таким образом, что экземпляр данного типа становится доступным для нее изнутри. В определение типа включается только заголовок метода. При определении метода он дополнительно идентифицируется именем типа. Поля и методы являются двумя составными частями новой структуры, называемой объектом. С учетом вышесказанного объект TPerson можно описать следующим образом:

```
type
  TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
    procedure Init(Nm,Dt: String; Rt: Real);
  end;
  procedure TPerson.Init(Nm,Dt:String; Rt:Real);
  begin
    Name:=Nm; {Поле Name объекта TStudent}
    Date:=Dt; {Поле Date объекта TStudent}
    Rate:=Rt; {Поле Rate объекта TStudent}
  end;
```

Таким образом, каждой операции соответствует метод — реализация этой операции для объектов данного класса. Можно сказать, что операция — это спецификация метода, а метод — реализация операции. Например, в классе «файл» может быть определена операция печати (print). Эта операция может быть реализована разными методами: печать двоичного файла; печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргумент — объект, к которому она применяется. Кроме того, операция может иметь и другие аргументы, иначе называемые параметрами. Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода. Метод связан только с классом и объектом.

Теперь для инициализации экземпляра типа TStudent достаточно просто вызвать его метод.

```
var
  Person : TPerson;
Person.Init('Николай Иванов' , '25-06-1995' , 400000);
```

Определение методов

Процесс определения методов объектов напоминает создание модулей в Turbo Pascal. Внутри объекта метод определяется заголовком процедуры или функции, действующей как метод:

```
type
TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
    procedure Init(Nm:Dt: String; Rt: Real);
    function GetName : String;
    function GetDate : String;
    function GetRate : Real;
end;
```

Поля данных должны быть объявлены перед объявлением методов. Подобно интерфейсной части модуля, описание методов внутри объекта только указывает действия, но не определяет, каким образом они будут выполнены. Сами методы описываются вне определения объекта как отдельная процедура или функция. При определении метода его имени должно предшествовать имя типа объекта, которому принадлежит данный метод, с последующей точкой:

```
procedure TPerson.Init(Nm, Dt: String; Rt: Real);
begin
    Name:=Nm;
    Date:=Dt;
    Rate:=Rt;
end;
function TPerson.GetName: String;
begin
    GetName :=Name;
end;
function TPerson.GetDate: String;
begin
    GetDate :=Date;
end;
function TPerson.GetRate: Real;
begin
    GetRate :=Rate;
end;
```

В программе при необходимости можно определить уже существующую функцию, например GetName, не связывая ее с типом TPerson.

Свойства объекта

Совокупность данных и методов их чтения и записи называется *свойством*. Свойства объектов можно устанавливать в процессе проектирования, а также можно изменять программно во время выполнения программы. В процес-

проектирования приложения в среде программирования Delphi можно просматривать значения некоторых из этих данных в окне Инспектора Объектов и изменять эти значения.

События и их обработка

Средой взаимодействия объектов являются *сообщения*, генерируемые в результате наступления различных *событий*.

ПРИМЕЧАНИЕ

Событие — это воздействие на объект. Список событий, на которые реагирует конкретный объект, можно посмотреть в Инспекторе Объектов на странице событий (Events).

События наступают в результате действий пользователя — перемещения курсора мыши, нажатия кнопок мыши или клавиш на клавиатуре, а также в результате работы самих объектов. В каждом объекте определено множество событий, на которые он может реагировать. В конкретных экземплярах объекта могут быть определены *обработчики* каких-то из этих событий, которые и определяют реакцию данного экземпляра объекта. К написанию этих обработчиков и сводится основное программирование при разработке графического интерфейса пользователя с помощью Delphi.

Итак, можно определить объект как совокупность свойств и методов, а также событий, на которые он может реагировать. Внешнее управление объектом осуществляется через обработчики событий. Эти обработчики обращаются к методам и свойствам объекта. Начальные значения данных объекта могут задаваться также в процессе проектирования установкой различных свойств. В результате выполнения методов объекта могут генерироваться новые события, воспринимаемые другими объектами программы или пользователем.

Создание и уничтожение объектов

Конечно, программа, спроектированная на основе ООП, — это не просто какая-то предопределенная совокупность объектов. В процессе работы объекты могут создаваться и уничтожаться. Таким образом, структура программы является динамическим образованием, меняющимся в процессе выполнения. Основная цель создания и уничтожения объектов — рациональное использование ресурсов компьютера, прежде всего, памяти. С целью организации динамического распределения памяти во все объекты заложены методы их создания — *конструкторы* — и уничтожения — *деструкторы*. Конструкторы объектов, которые изначально должны присутствовать в приложении, срабатывают при запуске программы. Деструкторы всех объектов, имеющихся в данный момент в приложении, срабатывают при завершении его работы. Но нередко и в процессе выполнения приложения различные новые объекты (например, новые окна документов) динамически создаются и уничтожаются с помощью их конструкторов и деструкторов.

Включать объекты в свою программу можно двумя способами: вручную включая в нее соответствующие операторы или путем визуального программирования, используя заготовки — *компоненты*.

Использование объектов при визуальном проектировании интерфейса

Как было сказано во введении, для преодоления трудностей на этапе создания интерфейса пользователя в конце XX века широкое распространение получило визуальное программирование, вооружившее программиста наглядными средствами конструирования интерфейса. Рассмотрим процесс визуального программирования на примере Delphi.

Работа производится в Интегрированной среде разработки (ИСП или Integrated Development Environment, IDE) Delphi. Среда предоставляет программисту *формы*, на которых размещаются компоненты. На форму с помощью мыши помещаются значки компонентов, имеющихся в библиотеке Delphi. С помощью простых манипуляций мышью можно изменять размеры и расположение этих компонентов. При этом в процессе проектирования можно постоянно видеть результат — изображение формы и расположенных на ней компонентов. Отпадает необходимость в запуске приложения для проверки того, как выглядит тот или иной компонент окна, и последующего изменения программного кода для подбора наиболее удачных размеров окна и других компонентов. Результаты проектирования можно видеть, даже не компилируя программу, немедленно после выполнения операции по изменению размеров и положения компонента с помощью мыши. Но достоинства визуального программирования не сводятся только к этому. Самое главное заключается в том, что во время проектирования формы и размещения на ней компонентов редактор кода Delphi автоматически генерирует код программы, включая в нее соответствующие фрагменты, описывающие данный компонент. А затем в соответствующих диалоговых окнах пользователь может изменить заданные по умолчанию значения свойств этих компонентов и, при необходимости, написать обработчики событий. То есть проектирование сводится, фактически, к размещению компонентов на форме, заданию некоторых свойств этих компонентов и написанию, при необходимости, обработчиков событий.

Компоненты могут быть *визуальными*, видимыми при работе приложения, и *невизуальными*, выполняющими некоторые служебные функции. Визуальные компоненты немедленно отображаются на экране в процессе проектирования в таком же виде, в каком их увидит пользователь во время выполнения приложения. Это позволяет очень легко выбрать место их расположения и дизайн — форму, размер, оформление, текст, цвет и т. д. Невизуальные компоненты отображаются на форме в процессе проектирования в виде значка, но пользователю во время выполнения они не видны, хотя и выполняют весьма полезную работу.

Как будет рассказано далее, в библиотеки визуальных компонентов Delphi включено множество типов компонентов и их число очень быстро растет от версии к версии. Имеющегося набора компонентов вполне достаточно, чтобы построить практически любое приложение, не прибегая к созданию новых компонентов.

Введение в Object Pascal

Возможности объектно-ориентированного проектирования в Delphi базируются на свойствах языка Object Pascal. Но ориентация приложений Delphi на Object Pascal несколько не сужает возможностей разработчика, так как приложения Delphi могут использовать разработки, сделанные с использованием других языков, включая C++ и даже ассемблер. Далее мы остановимся на том, как с помощью справочной системы Delphi получить подсказку по языку Object Pascal.

Нововведения в Object Pascal

Основные элементы языка Object Pascal ведут свое происхождение от Turbo Pascal и должны быть вам знакомы по первой части книги, поэтому рассмотрим некоторые новые элементы Object Pascal, появившиеся в Delphi 6.

Дополнительно к типам данных, имеющимся в Turbo Pascal, в Object Pascal введен новый тип данных *variant*. В переменных типа *variant* могут храниться данные любых типов, кроме записей, множеств, статических массивов, файлов, классов, ссылок на классы, указателей и *Int64*. Иначе говоря, переменные типа *variant* могут хранить все, кроме структур, указателей и *Int64*. В переменных типа *variant* могут храниться объекты COM и CORBA, а также динамические массивы и массивы *variant*. В Object Pascal Delphi 6 введена возможность создания собственных типов *variant*, допускающих перегрузку операций.

Новые функции в Object Pascal:

- функции сравнения — *CompareValue*, *EnsureRange*, *InRange*, *SameValue*;
- арифметические функции *DivMod*, *RoundTo*;
- функции работы с величинами *Bcd* (Binary-coded decimal) — *StrToBcd*, *TryStrToBcd*.
- ряд функций, работающих с Floating Point Unit (FPU) — словом состояния (SW), определяющим способ округления, точность вычислений и наличие прерываний или значений, определяемых константами *Infinity*, *NegInfinity*, *NaN* при вычислениях с плавающей запятой. Установкой и чтением FPU управляют функции *SetExceptionMask*, *SetPrecisionMode*, *SetRoundMode*, *GetExceptionMask*, *GetPrecisionMode*, *GetRoundMode* и некоторые другие. А проверка результатов на бесконечные и нечисловые значения осуществляют функции *IsInfinite* и *IsNaN*.

Добавлено множество новых функций всех категорий. Например, добавлено 150 новых функций работы с датами и временем, 45 функций работы с типом `variant` и т. д.

Общая организация программы в Delphi

Программа, создаваемая в среде Delphi в процессе проектирования приложения, основана на *модульном принципе*. Головная программа получается предельно простой и короткой. Она состоит из объявления списка используемых модулей и нескольких операторов, создающих объекты для необходимых форм и запускающих приложение на выполнение.

Модульность очень важна для создания надежных и относительно легко модифицируемых и сопровождаемых приложений. Четкое соблюдение принципов модульности в сочетании с принципом скрытия информации позволяет производить модификации внутри любого модуля, не затрагивая при этом остальных модулей и головную программу.

Все объекты компонентов размещаются в объектах — формах. Для каждой формы, проектируемой в приложении, Delphi создает отдельный модуль. Именно в модулях и осуществляется программирование задачи. В обработчиках событий объектов — форм и компонентов — размещаются описания алгоритмов, которые в основном сводятся к обработке информации, содержащейся в свойствах одних объектов, и задании по результатам этой обработки свойств других объектов.

Структура файла головной программы приложения Delphi

В процессе проектирования приложения Delphi автоматически создает код головной программы и отдельных модулей. В модули программист вводит свой код, создавая обработчики различных событий. Но головную программу, как правило, не приходится модифицировать и даже просматривать ее текст. Только в исключительных случаях бывает необходимо что-то изменить в тексте головной программы, сгенерированном Delphi. Головной файл приложения Delphi имеет следующую структуру:

```
program <имя>;
{Объявления подключаемых модулей, а также объявления локальных
 для головного файла типов, классов, констант, переменных,
 описания локальных функций и переменных}
begin
{Операторы тела программы}
end.
```

В Delphi головной файл обычно не содержит ничего, кроме операторов инициализации приложения, создания форм и запуска приложения. Вся логика программы помещается в файлы модулей.

Типичная головная программа приложения имеет следующий вид:

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2};
  {$R *.res}
  {Здесь вы можете поместить описания констант,
   переменных, функций, процедур, доступных
   для использования только в пределах данного файла}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

Программа начинается с ключевого слова `program`, после которого указывается имя программы. Оно совпадает с именем файла, в котором был сохранен проект. Это же имя присваивается исполняемому файлу приложения. По умолчанию используется имя `Project1`.

СОВЕТ

Всегда сохраняйте проект под осмысленным именем, изменяя тем самым имя проекта, заданное Delphi по умолчанию. Иначе очень скоро вы запутаетесь в программах `Project1`, лежащих в различных ваших каталогах.

После заголовка в тексте программы располагается предложение

```
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2};
```

В этом предложении перечисляются модули, загружаемые программой. Первый модуль `Forms` является системным, а следующие — модулями разработанных программистом форм. Данный пример подразумевает, что в проекте были созданы две формы с именами `Form1` и `Form2` в модулях с именами `Unit1` и `Unit2`. Заключенные в фигурные скобки названия форм представляют собой комментарий.

Предложение в головном файле программы может также содержать после имени модуля ключевое слово `in`, после которого указывается имя файла, в котором содержится модуль. Например:

```
uses Unit1 in 'Unit1.pas', Unit2 in 'c:\examples\Unit2.pas';
```

Ключевое слово `in` используется в случаях, когда необходимо указать Delphi, где следует искать соответствующие файлы.

ВНИМАНИЕ

В предложениях `uses`, включаемых в модули, использование `in` не допускается.

Следующая строка текста — `{R *.RES}` — представляет собой директиву компилятора, связывающую с исполняемым модулем файлы ресурсов. Она указывает файлы ресурсов (.DFM, .RES), которые должны быть включены в исполняемый модуль или библиотеку. Указанный файл должен быть файлом ресурсов Windows. По умолчанию для файлов ресурсов используется расширение .RES. В процессе компоновки программы или библиотеки файлы, указанные в директивах `{R}`, копируются в исполняемый модуль.

Первый оператор в теле программы `Application.Initialize`; инициализирует приложение, следующие за ним операторы `Application.CreateForm(TForm1, Form1);` и `Application.CreateForm(TForm2, Form2);` создают объекты форм `Form1` и `Form2`, последний оператор, `Application.Run`; начинает выполнение приложения.

Если требуется добавить какой-то текст в главную программу, то следует ввести описания необходимых констант, переменных, функций и процедур в место программы, отмеченное соответствующим комментарием в приведенном выше тексте. Кроме того, можно добавить или изменить операторы в теле программы, например, произвести настройки при запуске приложения на выполнение (скажем, настроить формы на русский язык) или сделать запрос пользовательского ввода и в зависимости от ответа создавать или не создавать те или иные формы. Например, если требуется, чтобы вторая форма приложения `Form2` создавалась только в случае, если при запуске приложения в командной строке указана опция `Y`, то следует заменить приведенный выше оператор `Application.CreateForm(TForm2, Form2);` на оператор

```
if(ParamStr(1) = 'Y')
    then Application.CreateForm(TForm2, Form2);
```

Этот оператор анализирует первый параметр командной строки. Если приложение `Project1` запущено командой `Project1 Y`, то будет создана форма `Form2`. В остальных случаях этой формы не будет.

ПРИМЕЧАНИЕ

Можно добавить в головной файл и другие операторы, функции, процедуры. Однако делать это не рекомендуется, поскольку такие действия противоречат принципу модульности. Все необходимые в начале выполнения процедуры и функции настройки следует помещать в отдельный модуль без формы.

Обратите внимание, что все определенные в головном файле приложения константы, переменные, процедуры, функции, типы будут доступны только в пределах этого головного файла и недоступны в модулях.

Общая структура модуля

Каждый модуль в общем случае имеет следующую структуру:

```
unit <имя модуля>;
interface // Открытый интерфейс модуля
{
    Сюда могут помещаться списки подключаемых модулей,
    объявления типов, констант, переменных, функций и
    процедур, к которым будет доступ из других модулей
}
```

```
implementation // Реализация модуля
{
    Сюда могут помещаться списки подключаемых модулей,
    объявления типов, констант, переменных, к которым
    не будет доступа из других модулей. Тут же должны быть
    реализации всех объявленных в разделе interface функций и
    процедур, а также могут быть реализации любых
    дополнительных, не объявленных ранее функций и процедур.
}
initialization {необязательный}
<Операторы, выполняемые один раз при первом обращении
к модулю>
finalization {необязательный}
<Операторы, выполняемые при любом завершении работы модуля>
end.
```

Рассмотрим теперь, как выглядят тексты модулей. Ниже приведен текст модуля с пустой формой. Подробные комментарии в этом тексте поясняют, куда и что в этот код можно добавлять.

```
unit Unit1
interface // Открытый интерфейс модуля
uses {Список подключаемых модулей}
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
    Forms, Dialogs;
type {Объявление класса формы}
    TForm1 = class(TForm)
    private // Закрытый раздел класса
    { Private declarations }
    {Объявления переменных, функций и процедур, включаемых в класс формы,
    но недоступных для других модулей}
    public // Открытый раздел класса
    { Public declarations }
    {Объявления переменных, функций и процедур, включаемых в класс формы и доступных
    для других модулей}
    end;
var
    Form1: TForm1;
{Объявления типов констант, переменных, функций и процедур, к которым будет
доступ из других модулей, но которые не включаются в класс формы}
implementation // Реализация модуля
{
    Сюда могут помещаться предложения uses, объявления типов, констант, переменных,
    к которым не будет доступа из других модулей. Тут же должны быть реализации всех
    объявленных в разделе interface функций и процедур, а также могут быть реализации
    любых дополнительных, не объявленных ранее функций и процедур.
}
{$R *.dfm}
end.
```

Модуль начинается с ключевого слова `unit`, после которого указывается имя модуля. Оно совпадает с именем файла, в котором был сохранен модуль.

По умолчанию для первого модуля используется имя Unit1, для второго — Unit2 и т. д.

СОВЕТ

Если в вашем приложении несколько модулей, сохраняйте их файлы с осмысленными именами, изменяя тем самым имена модулей, заданные Delphi по умолчанию.

Текст модуля состоит из двух основных разделов: *interface* — *открытый интерфейс* модуля, и *implementation* — *реализация* модуля. Все, что помещается непосредственно в раздел *interface* (типы, переменные, константы, функции, процедуры), может быть использовано другими модулями программы. Все, что помещается в раздел *implementation*, является внутренним делом модуля. Внешние модули не могут видеть типы, переменные, константы, функции и процедуры, размещенные в разделе реализации.

В разделе *interface* после предложения *uses*, содержащего список подключаемых модулей, располагается заготовка объявления класса формы, подготовленная Delphi. Имя класса формы — TForm1. Класс содержит два раздела: *private* — *закрытый* раздел класса, и *public* — *открытый* раздел класса. То, что объявлено в разделе *public*, будет доступно для других классов и модулей. То, что объявлено в разделе *private*, доступно только в пределах данного модуля.

За объявлением класса формы следуют строки:

```
var
Form1: TForm1;
```

Таким образом объявляется переменная Form1 класса TForm1.

Затем следует пока пустой раздел реализации *implementation*, в котором содержится только директива компилятора {\$R *.dfm}, обеспечивающая компоновку файлов ресурсов форм.

ВНИМАНИЕ

Директиву {\$R *.dfm} нельзя удалять из текста модуля, так как в противном случае загрузочный модуль не будет создан и будет сгенерировано исключение EResNotFound.

ПРИМЕЧАНИЕ

Кроме разделов *interface* и *implementation*, можно добавить в модуль еще два раздела: *initialization* и *finalization*.

Области видимости и доступ к объектам, переменным и функциям модуля

Структура модуля, содержащего объекты и процедуры

Рассмотрим текст кода модуля, обрабатывающего размещенные на форме два компонента: кнопка Button1 типа TButton и метка Label1 типа TLabel. Кроме того, в модуле реализован обработчик события, связанного со щелчком

мышью на кнопке, и в разных местах модуля добавлены переменные и функции, чтобы можно было видеть, как получить к ним доступ.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    procedure F1(Ch:char); {Процедура F1 доступна только в данном модуле}
  public
    {Переменная Ch1 и процедура F2 доступны для объектов любых классов
    и для других модулей, но со ссылкой на объект}
    Ch1:char;
    procedure F2(Ch:char);
  end;
var
  Form1: TForm1;
  {Переменная Ch2 и процедура F3 доступны для объектов любых классов
  и для других модулей}
  Ch2:char;
  procedure F3(Ch:char);
implementation
{$R *.DFM}
uses unit2;
{Переменная Ch3 и процедура F4 доступны только внутри данного модуля}
var Ch3:char;
  procedure F4(Ch:char);
begin
  Form1.Label1.Caption:=Form1.Label1.Caption+Ch+Form1.Ch1;
end;
  procedure TForm1.F1(Ch:char);
begin
  Label1.Caption:=Label1.Caption+Ch+Ch1;
end;
  procedure TForm1.F2(Ch:char);
begin
  Label1.Caption:=Label1.Caption+Ch+Ch1;
end;
  procedure F3(Ch:char);
begin
  Form1.Label1.Caption:=Form1.Label1.Caption+Ch+Form1.Ch1;
end;
```

```

procedure TForm1.Button1Click(Sender: TObject);
(Переменная Ch4 и процедура F5 доступны только внутри данной процедуры)
    var Ch4:char;
procedure F5(Ch:char):
begin
    Label1.Caption :=Label1.Caption+Ch+Ch1;
end;
begin
    Ch1:='-';
    Ch2:='A';
    Ch3:='B';
    Ch4:='C';
    Label1.Caption:='';
    F1(Ch1);
    F2(Ch2);
    F3(Ch3);
    F4(Ch4);
    F5('D');
    Label1.Font.Color:=clRed;
end;
end.

```

Структура программы является стандартной, но в текст внесены некоторые добавления. В описании класса можно видеть строки:

```

Label1: TLabel;
Button1: TButton;
procedure Button1Click(Sender: TObject);

```

Первые две были вставлены редактором кода Delphi автоматически, как только разработчик приложения разместил на форме метку и кнопку. Эти строки означают, что в форме имеются объекты соответствующих типов. Третью из приведенных строк редактор кода Delphi добавил в тот момент, когда разработчик приложения начал реализовывать обработку события `OnClick` — щелчка на кнопке. Одновременно с записью в описание класса объявления процедуры Delphi занесла в раздел `implementation` заготовку соответствующей процедуры.

```

procedure TForm1.Button1Click(Sender: TObject);
begin
end;

```

Здесь между резервированными словами `begin` и `end`; можно разместить текст обработчика события.

Помимо этой процедуры в коде присутствует еще несколько одинаковых процедур: `F1–F5` и несколько переменных символьного типа: `Ch1–Ch4`. Рассмотрим работу процедур `F1–F5`. У компонента — метки типа `TLabel` — имеется свойство `Caption` — надпись на метке. Каждая из процедур `F1–F5` берет значение этой надписи, добавляет к ней символ, переданный в нее в качестве параметра `Ch`

затем добавляет символ, хранящийся в переменной `Ch1`, и возвращает надпись с этими добавлениями обратно в метку.

Обработчик щелчка мышью на кнопке — процедура `Tform1.Button1Click` — задает символьным переменным `Ch1–Ch4` значения символов «-», «A», «B» и «C», затем очищает свойство `Caption` метки `Label1`, занося в него пустую строку, а затем поочередно обращается к процедурам `F1–F5`, передавая в них в качестве параметров различные символы. В заключение надпись метки окрашивается в красный цвет. Для этого используется свойство `Font` — шрифт объекта `Label1`. Это свойство само является объектом, имеющим свойство `Color` — цвет. Значение этого свойства изменяет последний оператор процедуры `TForm1.Button1Click`.

Разберемся, как обращаться из процедур к различным объектам, их свойствам и методам.

Доступ к свойствам и методам объектов

Для получения из программы доступа к свойствам и методам объекта следует указать ссылку на него в следующем формате:

<имя объекта> <имя свойства>

За именем объекта следует без пробела символ точки, а затем, тоже без пробела, — имя свойства. Например, ссылка на свойство `Caption` метки `Label1` осуществляется при помощи записи `Label1.Caption`.

Иногда свойство объекта в свою очередь является объектом. Тогда при обращении к этому свойству указывается вся цепочка предшествующих объектов, разделенных точкой. Например, метки имеют свойство `Font` — шрифт, которое в свою очередь является объектом. У этого объекта имеется множество свойств, в частности, свойство `Color` — цвет шрифта. Чтобы сослаться на цвет шрифта метки `Label1`, надо записать `Label1.Font.Color`, что означает «свойство `Color` объекта `Font`, принадлежащего объекту `Label1`». Аналогичная нотация с точкой используется и для доступа к методам объекта. Например, для метки, как и для большинства других объектов, определен метод `Free`, который уничтожает метку и освобождает занимаемую ею память. Если в какой-то момент потребовалось уничтожить метку `Label1` в приложении, то следует использовать оператор

```
Label1.Free;
```

ВНИМАНИЕ

Имейте в виду, что если по ошибке далее в программе будет выполняться оператор, обращающийся к свойствам или методам уничтоженного объекта, то это вызовет прерывание программы, поскольку этого объекта уже не будет в памяти. Обращение к объекту, удаленному из памяти методами `Destroy` или `Free`, вызывает ошибку и аварийное завершение выполнения программы.

Различия между переменными, функциями и процедурами, включенными и не включенными в описание класса

Теперь посмотрим, чем отличаются константы, переменные, функции и процедуры, включенные и не включенные в описание класса. В приведенном выше примере переменная Ch1 и процедуры F1 и F2 включены в описание класса, а переменные Ch2, Ch3 и процедуры F3 и F4 объявлены вне класса. Различие в их использовании будет заключаться в следующем.

- Для процедур, объявленных в классе, в их описании в разделе `implementation` к имени процедуры должна добавляться ссылка на класс. В нашем примере имена процедур F1, F2 и `Button1Click` в описании этих процедур в разделе `implementation` заменяются на `TForm1.F1`, `TForm1.F2` и `TForm1.Button1Click`.
- К переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, — с указанием имени объекта класса.
- Если в приложении создается только один объект данного класса (в нашем примере — только один объект формы класса `TForm1`), то различие, в основном, чисто внешнее.
- Обращение к переменным и процедурам, описанным внутри и вне класса, из процедур, описанных вне класса, выполняется следующим образом: к переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, — с указанием имени объекта класса. Поэтому в нашем примере в процедурах F3 и F4 обращение к переменной Ch1 имеет вид `Form1.Ch1`. По той же причине и обращение к свойству `Caption` объекта `Label1` в этих процедурах имеет вид `Form1.Label1.Caption`. Внешние по отношению к классу процедуры могут получить доступ к переменным и процедурам, объявленным в классе, только через ссылку на объект `Form1`.

ПРИМЕЧАНИЕ

Необходимость добавления к именам процедур, описанных в классе, ссылок на класс объясняется просто. Вне класса можно описать другую процедуру с тем же именем, например, F1, что и у процедуры класса. И тогда из процедур, не описанных в классе, можно будет ссылаться на обе эти процедуры F1, только на одну из них непосредственно по имени F1, а на другую через объект класса — `Form1.F1`. Благодаря этому, при описании процедур вне класса можно даже не знать имен всех процедур, описанных в классе, а сам этот класс может быть описан в другом модуле, текст которого недоступен, но путаницы при этом не возникнет.

Все эти ссылки на объект не требуются для процедур, объявленных в классе. Поэтому в процедурах `TForm1.F1`, `TForm1.F2` и `TForm1.Button1Click` ссылки на переменную Ch1 и на объект `Label1` не содержат дополнительных ссылок на объект формы.

Если в приложении создается несколько объектов одного класса, например, несколько форм класса `TForm1` (как мы увидим, это часто делается в приложениях с интерфейсом MDI), то проявляются более принципиальные различия между переменными, описанными внутри и вне класса. Переменные, описанные вне класса (в нашем примере это Ch2 и Ch3), так и остаются в одном экземпляре, а переменные, описанные в классе (в нашем примере Ch1), тиражируются столько раз, сколько объектов этого класса будет создано; т. е. в каждом объекте класса `TForm1` будет своя переменная Ch1 и все они никак не будут связаны друг с другом. Таким образом, в переменную, описанную внутри класса, можно заносить какую-то информацию, индивидуальную для каждого объекта данного класса. А переменная, описанная в модуле вне описания класса, может хранить только одно значение.

Области видимости переменных и функций

Теперь остановимся на вопросе об областях видимости элементов программы — констант, переменных, функций и процедур, т. е. о связи места их объявления в программе и места их использования. Частично мы уже затрагивали этот вопрос в предыдущем разделе, не упоминая о самом понятии области видимости.

Видимость отдельных элементов модуля, описанного в разделе, поясняется подробными комментариями в тексте этого модуля.

- Элементы, объявленные в разделе `interface` модуля вне описания типа, видимы и доступны внутри данного модуля и из внешних модулей. В рассмотренном примере это относится к переменной Ch2 и процедуре F3.
- Элементы, объявленные в разделе `implementation` модуля, видимы и доступны внутри данного модуля, но недоступны из внешних модулей. В рассмотренном примере это относится к переменной Ch3 и процедуре F4.
- Элементы, объявленные в классе в разделе `private`, видимы и доступны только внутри данного модуля. При этом из процедур, объявленных внутри класса, к ним можно обращаться непосредственно по имени, а из других процедур — только со ссылкой на объект данного класса. В рассмотренном примере это относится к процедуре F1. Если в модуле описано несколько классов, то объекты этих классов взаимно видят элементы, описанные в их разделах `private`.
- Элементы, объявленные в классе в разделе `public`, видимы и доступны для объектов любых классов и для других модулей. При этом из объектов того же класса к ним можно обращаться непосредственно по имени, а из других объектов и процедур — только со ссылкой на объект данного класса. В рассмотренном примере это относится к переменной Ch1 и процедуре F2.
- В классах, помимо обсуждавшихся ранее, могут быть еще разделы `protected` (защищенные). Элементы, объявленные в классе в разделе `protected`, види-

мы и доступны для любых объектов внутри данного модуля, а также для объектов классов — наследников данного класса в других модулях. Объекты из других модулей, классы которых не являются наследниками данного класса, не видят защищенных элементов.

- Элементы, объявленные внутри другой процедуры (в рассмотренном примере это переменная Ch4 и процедура F5, описанные внутри процедуры TForm1.Button1Click), являются локальными, т. е. они видимы и доступны только внутри данной процедуры или внутри процедур, вложенных в данную. При этом время жизни переменных, объявленных внутри процедуры, определяется временем выполнения этой процедуры. Так, переменная Ch4 в нашем примере создается в момент вызова процедуры TForm1.Button1Click и уничтожается при завершении работы этой процедуры. Она видима в самой процедуре TForm1.Button1Click и ее может видеть процедура F5. Вложенная процедура F5 доступна только из процедуры TForm1.Button1Click, в которой она описана.

Передача параметров в функции

В главе 5 мы подробно рассмотрели механизм передачи параметров при вызове процедур и функций в языке Turbo Pascal. В Object Pascal параметры в процедуры и функции могут передаваться в основном двумя способами: по значению и по ссылке. То, что вы видели в рассмотренном выше примере, — это передача параметра по значению. В этом случае в заголовке процедуры указывается имя параметра и его тип. Например:

```
procedure TForm1.F2(Ch:char);
```

В этом случае Ch — это локальное имя формального параметра, используемое только внутри данной процедуры. При вызове этой процедуры, который может иметь вид F2(Ch2); в памяти создается временная переменная с именем Ch и в нее копируется значение аргумента Ch2. После этого связь между Ch и Ch2 разрывается. Внутри процедуры можно изменять значение Ch, но это никак не отразится на значении внешней переменной Ch2, указанной в вызове функции в качестве аргумента.

Такая передача параметров по значению имеет свои достоинства и недостатки. Достоинство заключается в том, что процедура не может «испортить» переданный в нее аргумент. Это важно для надежной работы приложения и позволяет разрабатывать процедуру, не задумываясь о том, не использованы ли где-то в программе те же имена параметров. Но у передачи параметра по значению имеется и ряд недостатков. Во-первых, процедура не может изменить значение аргумента, а иногда это очень желательно. Во-вторых, копирование значения аргумента требует дополнительных затрат времени и памяти. И, в-третьих, после окончания работы процедуры временная переменная, хранившая значение параметра, уничтожается. Поэтому нельзя использовать, например, для накопления какой-то информации.

Для реализации второго способа передачи информации — по ссылке, перед именем параметра в заголовке функции должно быть указано ключевое слово var, например:

```
procedure TForm1.F2(var Ch:char);
```

В этом случае не происходит копирования значения аргумента в локальную временную переменную в процедуре. Процедура реально работает не с параметром, а со ссылкой — указателем на место хранения аргумента в памяти. И любые изменения параметра Ch, произведенные в процедуре, в действительности относятся не к этому параметру, а к тому аргументу, который передан при вызове процедуры. Таким образом, ключевое слово var позволяет возвращать информацию из процедуры в вызвавшую ее внешнюю процедуру.

Дальнейшее знакомство с языком объектно-ориентированного программирования Object Pascal мы продолжим в ходе изучения возможностей среды визуального объектно-ориентированного программирования Delphi 6 при проектировании приложений.

Приложения Windows

Среда разработки программ Delphi ориентирована, прежде всего, на создание программ для Windows. Приложение Windows является специальным типом программы для персонального компьютера, которая:

- должна иметь исполняемый файл специального формата;
- в общем случае работать в прямоугольном окне на экране;
- должна быть способна работать одновременно с другими приложениями Windows, включая свои собственные версии;
- уметь осуществлять связь и обмен данными с другими приложениями Windows.

Управляемая событиями архитектура Windows-приложения

Среда Windows основана на архитектуре, управляемой событиями. Это означает, что любой осуществляемый пользователем ввод рассматривается как событие. Состоит ли это событие в нажатии кнопки на мыши или в нажатии клавиши на клавиатуре, для Windows это является событием, для которого генерируется соответствующее сообщение.

Независимая от аппаратуры графика

Windows унифицирует процесс вывода информации на экран и принтер и сводит его к единому модулю, называемому *интерфейсом графического устройства* (GDI), который обеспечивает общий интерфейс для всех программ

Windows. Более того, Windows предлагает драйверы для большинства стандартных графических адаптеров и принтеров, поэтому приложение Windows будет работать без каких-либо изменений на большинстве имеющихся в мире аппаратных средств.

Многозадачный режим

Windows позволяет пользователю одновременно запускать несколько приложений, устраняя необходимость в написании программ, которые после прерывания выполнения остаются резидентными. В Windows не просто реализован многозадачный режим работы, но и обеспечивается ряд возможностей межпроцессорного обмена, таких как буфер для временного хранения изображений и динамический обмен данными.

В Windows работа нескольких приложений организована таким образом, что каждое приложение вместо полного экрана использует одно или несколько окон. С точки зрения программиста это означает, что программа осуществляет вывод текста или графики не в конкретное место экрана, а в окно — некоторую область, ограниченную рамкой. Также приложение должно разделять имеющуюся свободную память с другими приложениями.

Управление памятью

В типичном сеансе работы с Windows несколько приложений запускается на выполнение и заканчивает свою работу по несколько раз, но при этом не происходит загрузки в память каждого приложения после предыдущего, так как в этом случае в системе скоро закончилась бы свободная память. Вместо этого Windows временно перемещает большую часть памяти приложения либо в другую область памяти либо на диск, для обеспечения работы других приложений или самой операционной системы.

Таким образом, приложение Windows должно соответствовать правилам Windows по динамическому управлению памятью и не использовать непосредственного доступа к конкретным адресам памяти. Одно из основных преимуществ менеджера памяти Windows состоит в возможности разделения скомпилированного кода между приложениями. Например, если пользователь запустит два экземпляра одного и того же приложения, то эти два приложения будут использовать один скомпилированный код в памяти. Аналогично, приложение может динамически загрузить модуль библиотеки, который может совместно использоваться различными приложениями.

Ресурсы

Описания интерфейсов устройств приложения Windows для пользователя: меню, блоки диалога, курсоры, значки, растровые изображения, строки и клавиши акселераторов называются *ресурсами*. В Windows имеется средство обработки этих описаний за пределами исходного кода приложения. Ресур-

сы приложения объединяются с откомпилированным исполняемым файлом до запуска приложения. Для уменьшения использования памяти приложение загружает свои ресурсы в память только тогда, когда они необходимы.

Выделение спецификации ресурса из исходного кода дает дополнительное преимущество: внешний вид приложения может быть изменен без модификации исходного кода программы. В действительности для модификации ресурсов приложения нет необходимости даже иметь его исходный код. Это значительно упрощает процесс настройки и локализации существующих приложений Windows.

В Delphi предлагается специальное средство для создания и адаптации ресурсов — Image Editor.

Многодокументный интерфейс

Многодокументный интерфейс (Multiple Document Interface, MDI) — это набор пользовательских соглашений относительно создания окон, которые содержат внутри себя дочерние окна. Примером MDI является интегрированная среда разработки Turbo Pascal 7.0.

ПРИМЕЧАНИЕ

Среда Delphi соответствует спецификации Single Document Interface (SDI) — однодокументный интерфейс, и состоит из нескольких отдельно расположенных окон. В Delphi вы можете одновременно работать только над одним проектом.

Как вы уже могли заметить, программирование в среде Windows является достаточно сложной задачей. Эта задача существенно облегчается благодаря объектно-ориентированному программированию, позволяющему разработчику приложения сосредоточиться на функциях приложения, а не на его форме. Используя объекты для представления сложных структур, подобных окнам, программа на Delphi может объединять свои операции и хранение данных.

Объектно-ориентированное программирование создает среду, в которой программист использует объекты для представления достаточно сложных элементов интерфейса пользователя программы Windows. Например, окно является объектом. Объектно-ориентированное окно и типы объектов приложения управляют обработкой сообщений, которая требуется от программы, что в значительной степени упрощает взаимодействие программиста и пользователя. В действительности объекты представляют собой не только окна, но и блоки диалога и управления (блоки списков и клавиши).

Автоматизация реакции системы в виде сообщений

Кроме инструкций для среды Windows о необходимости проведения определенных действий, большинство приложений должно иметь возможность реагировать на сотни сообщений Windows, которые являются результатом дей-

ствий пользователя (например, щелчок мышью), а также на сообщения от других приложений и прочих источников. Обработка сообщений и корректная реакция на них служит определяющим фактором правильной работы программы.

Основы единого графического интерфейса

Так как при разработке собственных проектов вам придется реализовывать средства взаимодействия пользователя с приложением, рассмотрим основные элементы, из которых состоит интерфейс пользователя, работающего в среде Microsoft Windows.

Базовым элементом интерфейса является окно. Окна составляют основу пользовательского интерфейса Microsoft Windows и являются основным средством взаимодействия пользователя с прикладной программой. Большинство остальных элементов интерфейса представляют собой частные случаи окна. *Окно* — это прямоугольная область экрана, выделяемая прикладной программой для осуществления операций ввода-вывода. Окна могут быть *перекрывающимися* (cascade) и *заполняющими экран* (tile). На рис. 11.2 показаны основные экранные компоненты приложения Windows.

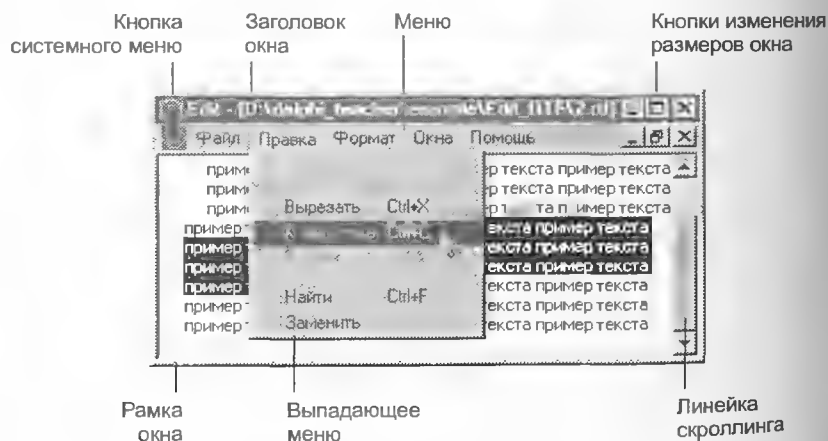


Рис. 11.2. Экранные компоненты приложения Windows

Заголовок окна — это специальная область, обычно содержащая строку текста и располагающаяся в верхней части окна. Заголовок может содержать кнопку системного меню и кнопки управления размерами окна. Системное меню — это меню, наличие которого в окне обозначается специальной кнопкой. В этом меню содержатся команды для перемещения, изменения размеров окна или его закрытия. Рамка — прямоугольная окантовка окна, используемая для указания границ окна и их изменения. Меню — набор команд, из которых можно сделать выбор с помощью клавиатуры или мыши. Резуль-

том выбора команды меню может быть либо выполнение какого-либо действия, либо появление вложенного меню, также содержащего команды. Полоса прокрутки — вертикальная или горизонтальная полоса с передвигающимся бегунком посередине и кнопками-стрелками по краям. Полосы прокрутки предназначены для позиционирования той части изображения, которая должна быть видна в окне. Окно может содержать в себе дочерние окна. При закрытии основного окна закрываются и его дочерние окна.

Специальный тип окна, используемый для отображения и ввода информации пользователем, называется *диалоговым окном*. В большинстве прикладных программ диалоговые окна связаны с определенными командами меню. Особенностью диалоговых окон является отсутствие кнопок изменения размеров. Обычно диалоговые окна содержат одну или несколько кнопок, позволяющих пользователю выполнить определенные действия. Часто в состав панели диалога включаются комбинированные списки, строки редактирования, группы кнопок и другие элементы управления.

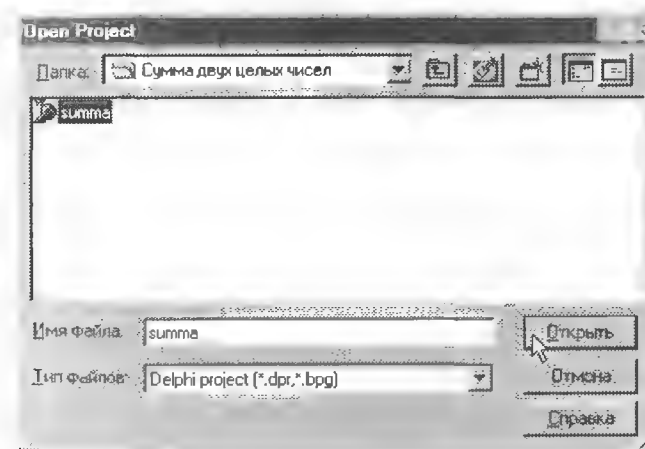


Рис. 11.3. Диалоговое окно

Диалоговые окна бывают модальными и немодальными. Наиболее часто используются модальные окна. В *модальном окне* для продолжения работы с другими окнами пользователь должен нажать на кнопку и закрыть это окно. Эти окна не дают пользователю возможности работать с другими окнами, созданными приложением, породившим диалоговое окно, но разрешают переключаться на работу с другими приложениями. Модальные окна отличаются отсутствием полос прокрутки, фиксированным размером и невозможностью минимизации (свертывания в значки). В особых случаях, представляющих угрозу системе и требующих немедленной реакции оператора, могут использоваться системные модальные окна. Эти окна блокируют работу пользователя с другими окнами вплоть до своего закрытия.

Немодальные диалоговые окна не требуют закрытия для продолжения работы, и пользователь может во время работы с ними свободно переключаться на любое приложение. Например, диалоговое окно Find, позволяющее искать строки символов, может быть немодальным.

Частным случаем модального диалогового окна является окно сообщений, используемое для вывода информации типа сообщения о статусе операции или об ошибке. Помимо сообщения такая панель может содержать одну или несколько кнопок, позволяющих пользователю указать свою реакцию на выводимое в панель сообщение. Чаще всего используются кнопки Ok, Cancel, Retry.

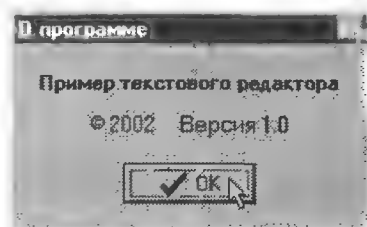


Рис. 11.4. Окно сообщения

Многие окна содержат списки команд, называемые *меню*. Различают строчное (горизонтальное) меню и вложенное (вертикальное). Глубина вложенности является довольно большой величиной и позволяет создавать комплексные меню. Обычно выбор команды меню приводит либо к выполнению каких-либо действий, либо к появлению вложенного меню или панели диалога.

Для управления окном часто используются кнопки. *Кнопка* — это элемент управления прямоугольной формы, имеющий название и посылающий соответствующее сообщение при нажатии. Обычно одна из букв названия кнопки выделяется специальным образом и служит для имитации нажатия кнопки с клавиатуры. Для выбора одного из нескольких элементов в диалоговом окне часто используются списки. *Список* — это набор элементов, один или несколько из которых могут быть выбраны. Типичным примером такого набора элементов может служить список имен файлов.

С остальными элементами окна мы познакомимся в процессе разработки приложений.

Архитектура Windows-программы

Программист, впервые взявшись за создание Windows-программы, обнаружит, что этот процесс существенно отличается от процесса создания программы для среды DOS. Архитектура Windows, управляемая событиями, накладывает отпечаток и на структуру самой прикладной программы. Для сравнения

представим программу стандартной архитектуры (рис. 11.4) и управляемой событиями (рис. 11.5).

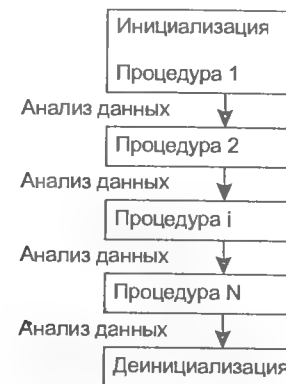


Рис. 11.5. Программа стандартной архитектуры

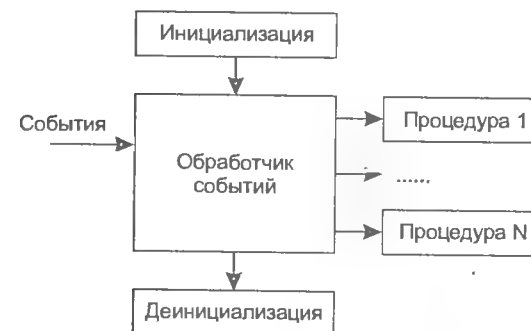


Рис. 11.6. Архитектура программы, управляемой событиями

Как видно из рис. 11.5, в программе стандартной архитектуры каждая процедура обрабатывает некоторые данные и в результате работы образуются новые данные, на основе которых производится выбор процедуры, которая должна выполняться следующей. В начале и в конце программы определяются действия по инициализации и деинициализации.

Из рис. 11.6 видно, что программа, управляемая событиями, не имеет привычных переходов от процедуры к процедуре. В этом случае сразу после выполнения инициализации начинает работать обработчик событий, который, производя анализ поступившего события, передает управление процедуре-обработчику события, а после обработки события управление возвращается системе. После этого обработчик готов к приему следующего события. Информация о наступившем событии передается в виде сообщения, которое раскрывает характер события и содержит информацию, необходимую для его обработки. По окончании работы программы производится деинициализация.

Таким образом, для программы, работающей в среде Windows, необходимо создать ряд процедур:

- процедуру инициализации;
- процедуру-обработчик событий;
- диспетчер событий;
- процедуры-обработчики конкретных событий;
- процедуру деинициализации.

Весь вывод информации на экран и прием сообщений Windows-программа осуществляет посредством окон, поэтому помимо работы с сообщениями необходимо выполнять работу с окнами.

Контрольные вопросы и задания

1. В чем заключается основное отличие процедурного и объектно-ориентированного программирования?
2. Что такое объект? Что общего имеет тип `object` с типом `record`? Чем они отличаются?
3. Что такое класс?
4. Как можно обратиться к полю объекта?
5. В чем заключается наследование?
6. Что такое инкапсуляция и для чего она применяется?
7. В чем заключается полиморфизм операций?
8. Что такое метод? Каковы особенности описания методов?
9. Что такое свойство объекта, каким образом можно его изменять?
10. Что такое события? Каково назначение обработчика события?
11. Каково назначение сообщений?
12. Что такое конструктор, деструктор?
13. В чем заключаются преимущества визуального проектирования интерфейса?
14. Опишите структуру и назначение отдельных элементов головной программы приложения Delphi.
15. Каково назначение модуля в проекте приложения Delphi? Опишите назначение отдельных разделов модуля.
16. Как различается доступность объектов, описанных в разделах `interface` и `implementation` модуля?
17. Как указать ссылку на свойства и методы объекта в тексте программы?
18. Каковы отличия в передаче параметров в процедуры и функции по значению и по ссылке?

глава 12

Интегрированная среда разработки Delphi 6

Назначение

Delphi — это потомок среды программирования Turbo Pascal. Название среды разработки приложений Delphi произошло от названия города в Древней Греции, где находился знаменитый Дельфийский оракул.

ПРИМЕЧАНИЕ

Дельфийский оракул — храм Аполлона в городе Дельфы, жрецы которого занимались предсказаниями.

Система визуального объектно-ориентированного проектирования Delphi позволяет:

- создавать законченные приложения для Windows самой различной направленности, от чисто вычислительных и логических, до использующих графику и мультимедиа;
- быстро создавать профессионально выглядящий оконный интерфейс для любых приложений, написанных на любом языке; интерфейс удовлетворяет всем требованиям Windows и автоматически настраивается на ту систему, которая установлена на компьютере пользователя, поскольку использует функции, процедуры и библиотеки Windows;
- создавать свои динамически присоединяемые библиотеки (DLL) компонентов, форм, функций, которые затем можно использовать из других языков программирования;
- создавать мощные системы работы с локальными и удаленными базами данных любых типов;
- формировать и печатать сложные отчеты, включающие таблицы, графики и т. п.;
- создавать справочные системы (файлы `.hlp`), как для своих приложений, так и для любых других, с которыми можно работать не только из приложений, но и просто из Windows;

- создавать профессиональные программы установки для приложений Windows, учитывающие всю специфику и все требования операционной системы.

Delphi — быстро развивающаяся система. Первая версия Delphi 1.0 была выпущена в феврале 1995 г. В 1996 г. вышла версия Delphi 2.0, в 1997 — Delphi 3.0, в 1998 — Delphi 4.0, в 1999 — Delphi 5.0, в 2001 — Delphi 6.0. Все версии, начиная с Delphi 2.0, рассчитаны на разработку 32-разрядных приложений, т. е. приложений для операционных систем Windows 95/98, Windows NT. В 2002 г. вышла версия Delphi 7, основные нововведения в которой связаны с интернет-технологиями.

Общее описание среды

Интегрированная среда разработки Delphi 6 (в дальнейшем будем называть ее кратко, ИСР) — это среда, в которой есть все необходимое для проектирования, запуска и тестирования создаваемых приложений: редактор исходного кода, отладчик, инструментальные панели, редактор изображений, инструментальный набор инструментов, гармонично дополняющих друг друга и облегчающих процесс создания современных приложений.

Большинство версий Delphi выпускается в нескольких вариантах:

- Standard — стандартный;
- Professional — профессиональная версия;
- Client/Server — клиент/сервер Enterprise — разработка баз данных предметных областей.

Эти варианты различаются, в основном, разным уровнем доступа к системам управления базами данных. Последние два варианта, Client/Server и Enterprise, являются наиболее мощными в этом отношении. В то же время библиотеки компонентов в различных вариантах практически одинаковы. Так что если ваши задачи не связаны с базами данных, то вы вполне можете обойтись стандартным вариантом. В Delphi 5–6 вариант Standard включает в качестве дополнительного инструментария только редактор изображений Image Editor. Вариант Professional включает также многочисленные инструменты, связанные с разработкой баз данных, за исключением специального построителя запросов SQL — SQL Builder, SQL Monitor — инструмента, позволяющего наблюдать в процессе отладки прохождение запросов, и TeamSource — инструмента, облегчающего планирование больших разработок группой проектировщиков. Вариант Enterprise включает в себя весь инструментарий.

Для запуска Delphi следует выбрать в Главном меню Windows команды Пуск ► Программы ► Borland Delphi 6 ► Delphi 6 или щелкнуть на ярлыке программы Delphi 6 на рабочем столе. После запуска Delphi на экране компьютера появляется основное окно интегрированной среды разработки, представленное на рис. 12.1.

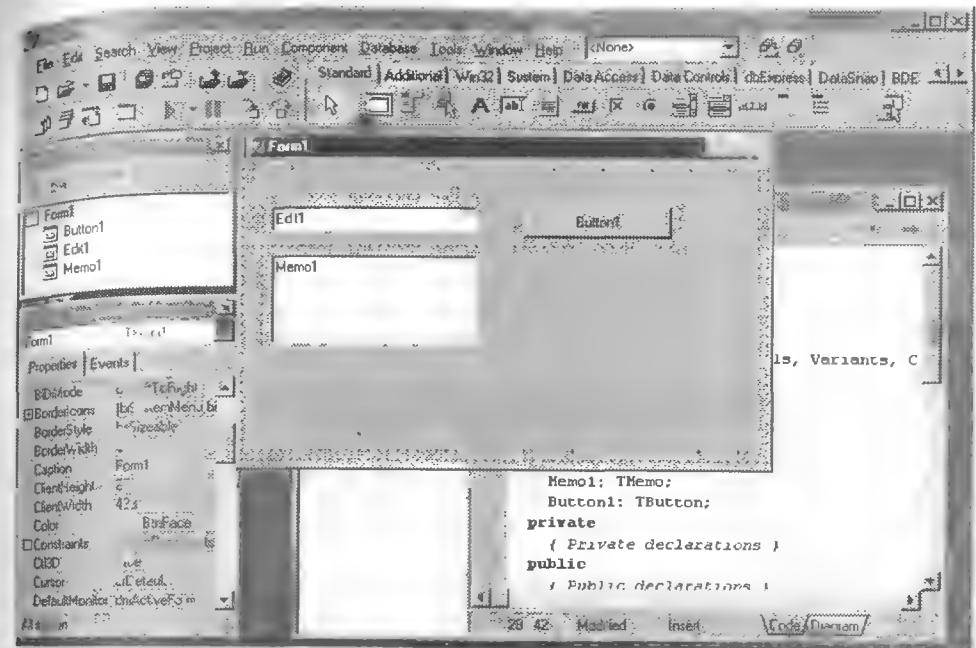


Рис. 12.1. Вид окна интегрированной среды разработки Delphi 6

В верхней части окна ИСР отображается полоса главного меню. Ниже полосы главного меню расположены две инструментальные панели. Левая панель (состоящая, в свою очередь, из трех панелей) содержит два ряда кнопок, дублирующих некоторые наиболее часто используемые команды меню (рис. 12.2).



Рис. 12.2. Панель кнопок, дублирующих команды меню с всплывающей подсказкой

Правая панель содержит панель библиотеки визуальных компонентов (Visual Component Library — VCL). В дальнейшем мы для краткости будем называть библиотеку визуальных компонентов просто палитрой. Палитра компонентов содержит ряд страниц, закладки которых видны в ее верхней части (рис. 12.3).

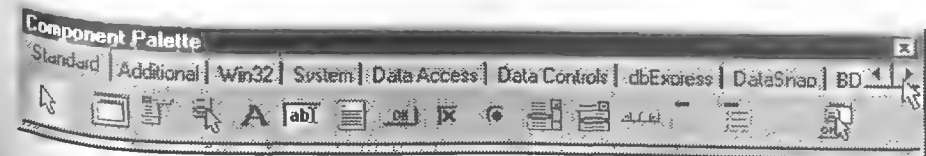


Рис. 12.3. Палитра компонентов

Правее полосы главного меню располагается еще одна небольшая инструментальная панель, содержащая раскрывающийся список и две кнопки.



Рис. 12.4. Панель выбора и сохранения различных конфигураций ИСР

Эта панель служит для сохранения и выбора различных конфигураций окна ИСР, которые вы сами можете создавать и запоминать (рис. 12.4).

Под палитрой компонентов располагается окно формы с размещенными на ней компонентами (рис. 12.5).

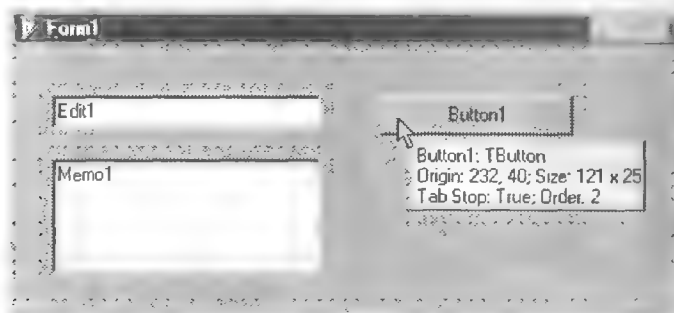


Рис. 12.5. Окно формы с размещенными компонентами

Форма, на которой размещаются другие компоненты, является основой почти всех приложений Delphi.

ПРИМЕЧАНИЯ

В начале создания нового проекта форма не содержит компонентов.

В некоторых случаях при разработке какого-то модуля форма может оказаться вообще ненужной.

Форму можно понимать как типичное окно Windows. Она обладает теми же свойствами, что и другие окна Windows: имеет управляющее меню в верхнем левом углу, полосу заголовка, занимающую верхнюю часть, кнопки разворачивания, сворачивания и закрытия окна в верхнем правом углу. Используя Инспектор объектов, можно изменить вид окна, убрав какие-то кнопки или всю полосу заголовка, сделав его окном с неизменяемыми размерами и т. п. Во время проектирования форма покрыта сеткой из точек. В узлах этой сетки размещаются те компоненты, которые разработчик помещает на форму. Во время выполнения приложения эта сетка не видна.

После того как на форме размещены какие-либо компоненты, можно получить по ним контекстную справку. Для этого следует выделить интересующий компонент и нажать клавишу F1. Если щелкнуть на самой форме и нажать F1, то будет показана справка по классу формы (рис. 12.6).

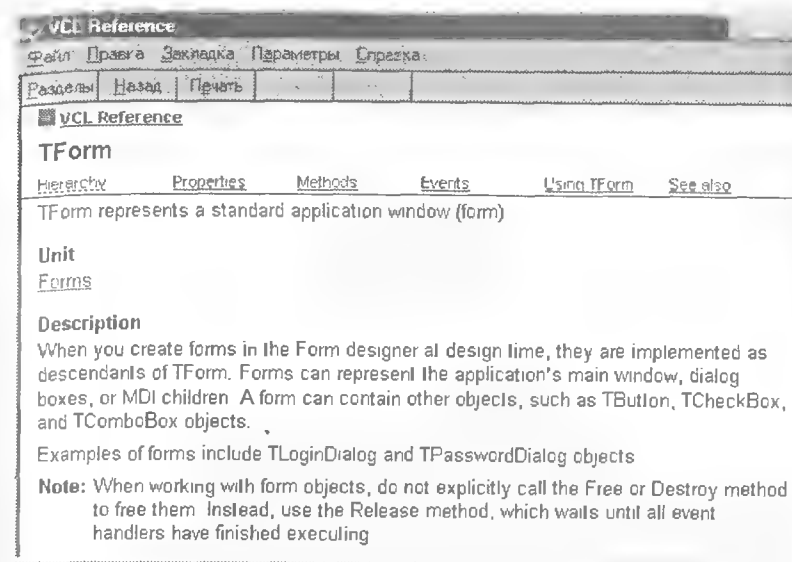


Рис. 12.6. Окно справки по форме

В основном поле окна слева находится окно Object Inspector (Инспектор объектов), с помощью которого в дальнейшем можно будет задать свойства компонентов и обработчики событий. На рис. 12.7. в окне Инспектора объектов отображается информация о свойствах формы Form1.

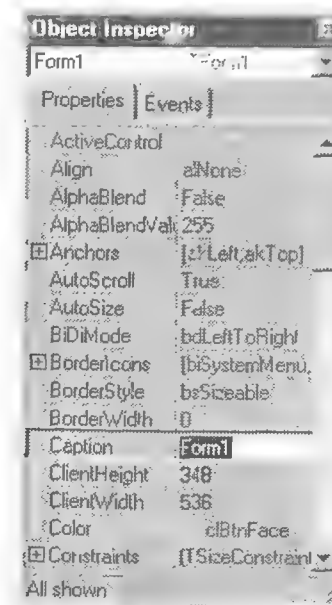


Рис. 12.7. Окно Инспектора объектов

Выше окна Object Inspector на рис. 12.1 расположено окно Object TreeView (Дерево объектов) (рис. 12.8), отображающее иерархию компонентов приложения с точки зрения их принадлежности друг другу. В дереве объектов можно осуществлять операции щелчка и перетаскивания, перемещая дочерние компоненты на другие контейнеры, при этом изменения синхронно отображаются в редакторе форм.

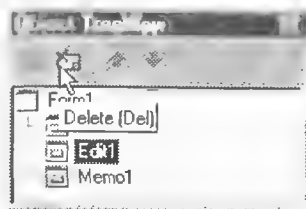


Рис. 12.8. Окно Дерева объектов

ПРИМЕЧАНИЕ

Отметим, что компоненты, у которых не определены ключевые свойства, отмечаются в дереве объектов вопросительными знаками, сразу же привлекающими внимание. Кроме того, дерево объектов отображает и компоненты, создаваемые неявно.

Ниже окна формы расположено окно Code Editor (Редактор кода) (рис. 12.9). Обычно оно при первом взгляде на экран невидимо, так как его размер равен размеру формы и окно Редактора кода практически полностью перекрывается окном формы.

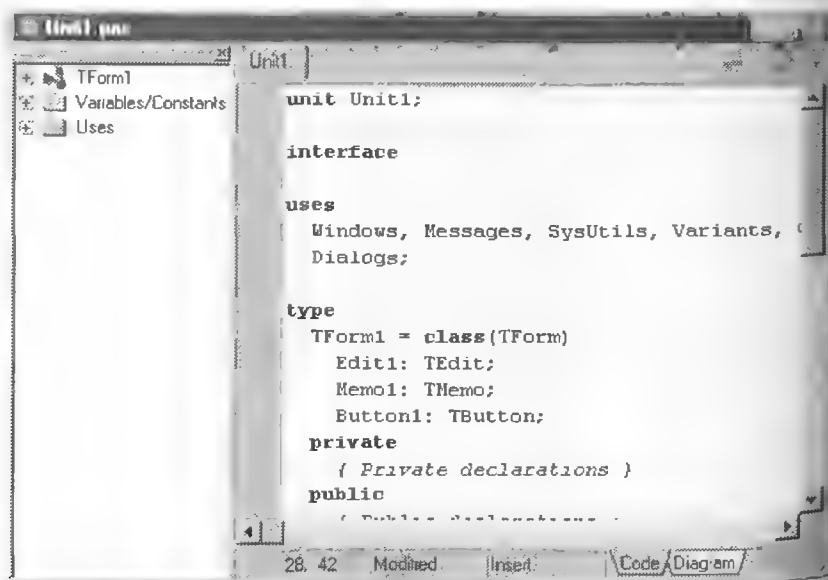


Рис. 12.9. Окно Редактора кода

На рис. 12.1 это окно немного сдвинуто вправо и вниз и «выглядывает» из-под окна формы.

Подробное описание элементов окна интегрированной системы программирования Delphi 6 можно найти в приложении В.

Одним из способов работы с ИСР является выбор команды из меню. Выбор команды меню выполняется любым из стандартных способов: нажатием F10, Alt в сочетании с «горячей клавишей» или щелчком мыши по соответствующему пункту меню. Можно уточнить назначение пункта меню, используя справочную систему Delphi. Для получения справки Delphi о пункте меню следует выбрать интересующий вас пункт меню, например, команду Run в меню Run и нажать клавишу F1. После этого откроется окно Delphi Help, содержащее справочную информацию, как показано на рис. 12.10.

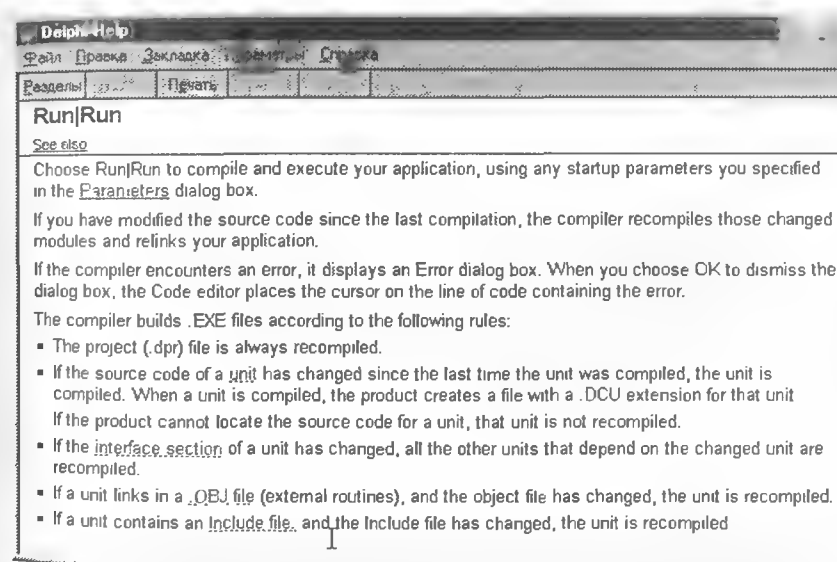


Рис. 12.10. Окно Delphi Help со справочной информацией о пункте меню

Создание, компиляция и отладка простого приложения

Основные операции в интегрированной среде разработки Delphi рассмотрим на примере создания, компиляции и отладки простого приложения.

Упражнение 1. В первой главе книги была разобрана задача вычисления суммы двух целых чисел. Создадим в ИСР Delphi приложение, которое обеспечивает ввод двух целых чисел, по щелчку на кнопке с символом «=» вычисляет их сумму и выводит значение результата.

Создание формы с размещением визуальных компонентов

После запуска ИСР Delphi создайте новый проект при помощи команды File ► New ► Application. В результате создания проекта приложения в окне Delphi будет раскрыто окно формы, на которой можно размещать визуальные компоненты проекта. Сохраните новый проект, для чего выберите команду File ► Save Project As. В окне Save Unit1 As создайте новую папку для файлов создаваемого проекта, например, «Сумма двух целых чисел», как показано на рис. 12.11.

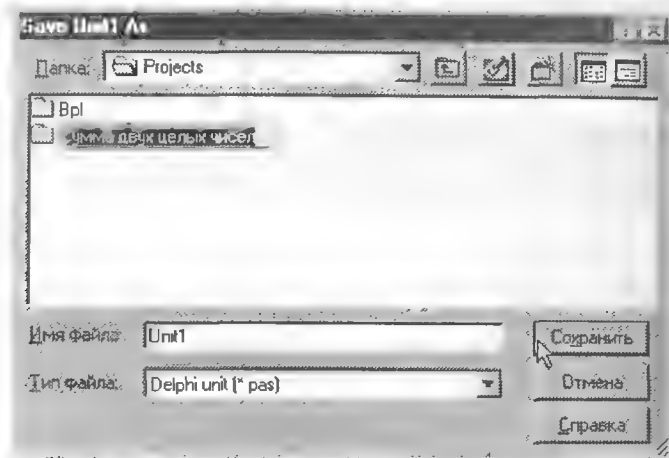


Рис. 12.11. Создание папки для нового проекта

Затем в окне Save Unit1 As откройте созданную папку и нажмите кнопку Сохранить. После сохранения файла модуля Unit1.pas откроется окно Save Project As (рис. 12.12). Задайте имя файла проекта, например, «summa» и нажмите кнопку Сохранить.

Измените свойства формы. Для изменения размеров формы захватите угол окна формы и, не отпуская левую кнопку мыши, перемещайте мышь, задавая требуемый размер формы. Измените надпись в заголовке формы Form1 с помощью Инспектора объектов. Инспектор объектов обеспечивает простой и удобный интерфейс для изменения свойств объектов Delphi и управления событиями, на которые реагирует объект. Окно Инспектора объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения компонента. Первая страница называется Properties (Свойства), вторая — Events (События). Над ними располагается раскрывающийся список всех компонентов, размещенных на форме. В нем можно выбрать тот компонент, свойства и события которого вас интересуют. Страница Properties Инспектора объектов (рис. 12.13), показывает свойства того объекта, который в данный момент выделен. Для переключения между страницами свойств

и событий используются закладки Properties и Events в верхней части окна Инспектора объектов.

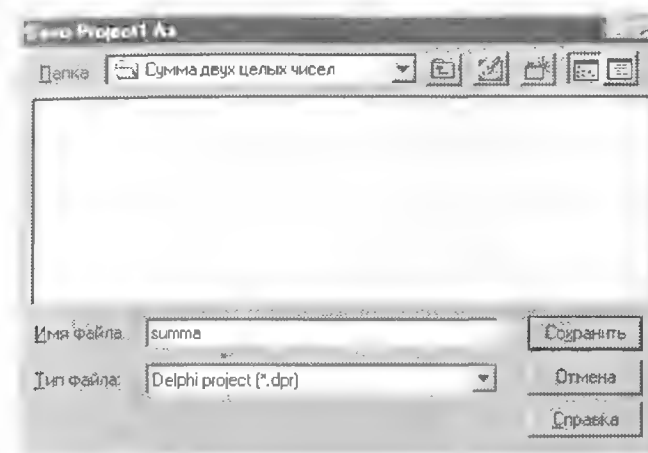


Рис. 12.12. Сохранение проекта

При щелчке на некоторых свойствах, например, на свойстве Color, справа от имени свойства открывается окно раскрывающегося списка. Нажав в нем кнопку со стрелкой вниз, можно увидеть список возможных значений этого свойства. Список значений можно просмотреть с помощью бегунка. Например, если изменить значение свойства Color с принятого по умолчанию clBtnFace (цвет поверхности кнопок) на clWmdow (цвет окна), поверхность формы изменит свой цвет.

Рядом с некоторыми свойствами можно заметить знак плюс (см., например, свойство Font на рис. 12.16). Это означает, что данное свойство является объектом, который в свою очередь имеет ряд свойств. После щелчка на этом плюсе или двойного щелчка на свойстве Font откроется таблица таких свойств, как Color (цвет), Height (высота), Name (имя шрифта) и др. Среди них есть свойство Style (стиль), около которого тоже имеется знак плюса. Щелчок на этом плюсе или двойной щелчок на этом свойстве раскроет дополнительный список подсвойств, в котором можно, например, установить в значение true для свойства fsBold (жирный). Кстати, для смены true на false и обратно в подобных булевых свойствах не обязательно выбирать значение из раскрывающегося списка. Для его изменения достаточно двойного щелчка на значении свойства. После просмотра или изменения подсвойств можно опять произвести двойной щелчок на головном свойстве или щелчок на знаке минус около него, и список подсвойств будет закрыт.

Для изменения заголовка формы Form1 в окне Инспектора объектов откройте страницу Properties (Свойства), затем выберите свойство Caption (Заголовок) и задайте новое значение этого свойства, например, «Сумма двух целых чисел», как показано на рис. 12.13.

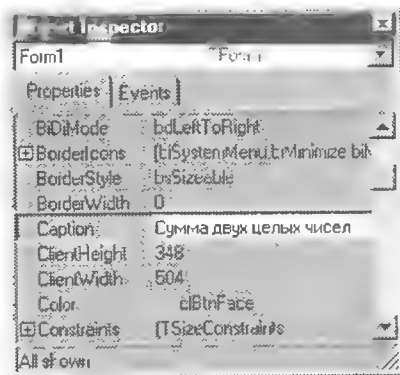


Рис. 12.13. Изменение свойства Caption формы Form1

Разместите на форме компоненты Edit1, Edit2, Edit3, Label1, Button1, как показано на рис. 12.14.

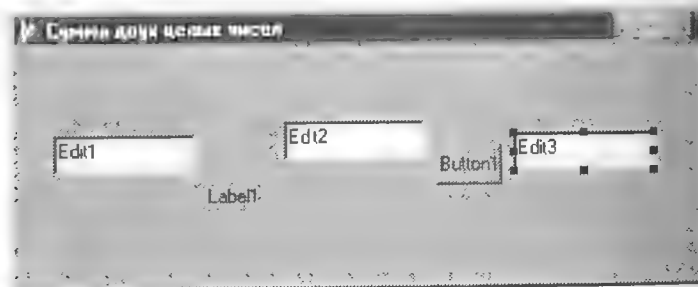


Рис. 12.14. Форма с размещенными на ней компонентами

Для размещения компонентов на форме, щелкнув на вкладке Standard, откройте палитру с требуемыми компонентами, затем, щелкнув на значке соответствующего компонента, например, Edit, щелкните в окне формы. Задайте положение и размер компонента при помощи мыши.

СОВЕТ

Если вы забыли, на какой странице палитры расположен конкретный компонент, выберите команду View ► Component List и на экране появится список компонентов в алфавитном порядке. Выбрав в окне Components нужный компонент, нажмите кнопку Add to form для размещения компонента на форме.

ПРИМЕЧАНИЕ

Есть и другой способ поместить компонент на форму — достаточно сделать двойной щелчок на значке компонента, и он будет автоматически помещен в центр формы. Если вы выбрали компонент, а затем решили не размещать его, достаточно нажать кнопку указателя. Это прервет процесс размещения компонента и программа вернется в нормальный режим, в котором можно выбрать другой компонент или выполнить какую-либо команду.

Получение справки о назначении компонентов

Имена компонентов, соответствующих значкам на палитре, можно узнать из ярлычка, появляющегося, если задержать над этим значком курсор мыши.

ПРИМЕЧАНИЕ

Имена на ярлычках выглядят так: MainMenu, Button и т. д. Однако в Delphi все имена классов в действительности начинаются с символа «Т», например, TMainMenu, TButton. Под такими именами во встроенной в Delphi справочной системе можно найти описания соответствующих компонентов.

Назначение компонентов можно уточнить, используя систему контекстной помощи Delphi. Для этого следует выделить нужную страницу в палитре компонентов и нажать клавишу F1. После этого откроется окно Delphi Help со справочной информацией. Например, окно справки о компонентах страницы Standard выглядит следующим образом:

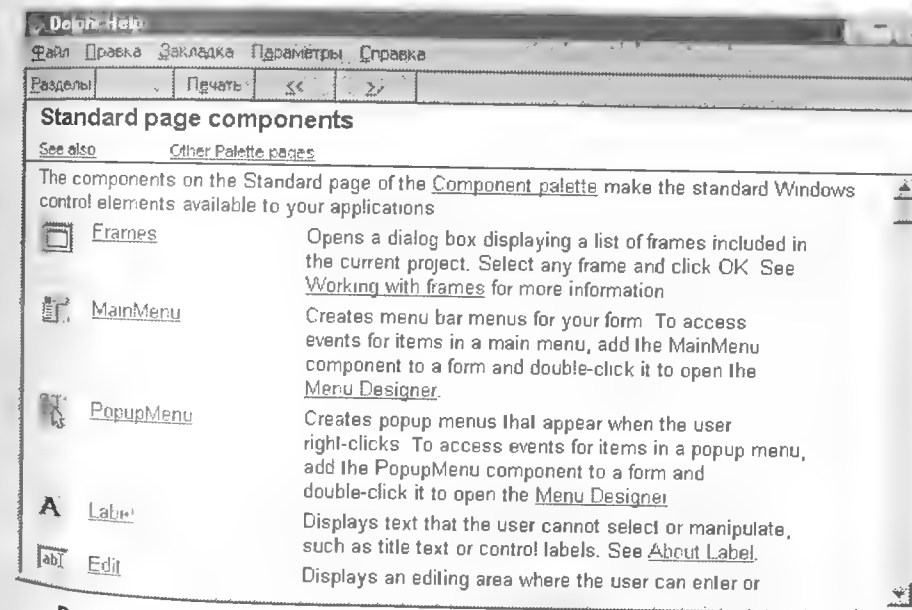


Рис. 12.15. Окно справочной информации о компонентах страницы Standard

Для просмотра информации в окне используется полоса прокрутки. Если требуется просмотреть справку о конкретном компоненте, то следует выбрать нужную ссылку. Если выбрать в палитре компонент и нажать F1, то будет показана справка по типу данного компонента. Нажатие кнопки Закрыть в окне Delphi Help приводит к закрытию окна со справочной информацией. Задайте свойство Caption компонента Label1 «+». Измените размер символов компонента Label1, для чего в окне Инспектора объектов выберите в списке объект Label1, затем на странице Properties разверните список свойств Font.

В списке свойств Font выберите свойство Size и задайте для него значение 20, как показано на рис. 12.16.

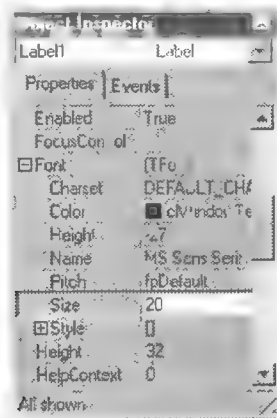


Рис. 12.16. Изменение значения свойства Font.Size

Аналогичным образом задайте свойство Caption компонента Button1 «=» и размер символов, равный 20.

ПРИМЕЧАНИЕ

Помимо главного меню в Delphi имеется система контекстных раскрывающихся меню, которые появляются при щелчке правой кнопкой мыши на каком-либо компоненте, как показано на рис. 12.17. Большинство разделов этих контекстных меню дублируют основные разделы главного меню.

Для выравнивания компонентов на форме при нажатой клавише Shift выделите компоненты и правой кнопкой мыши выберите в контекстном меню команду Position ► Align.

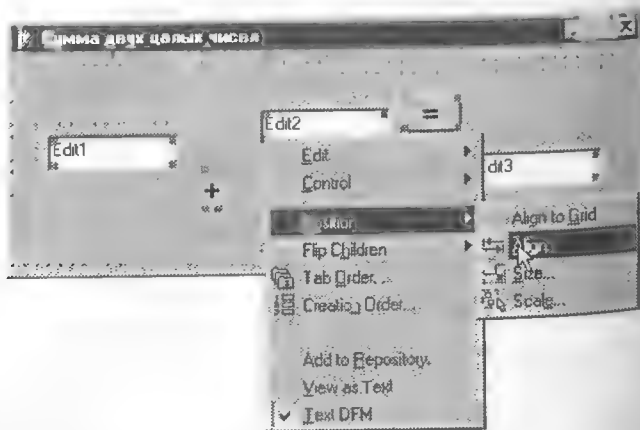


Рис. 12.17. Выравнивание компонентов

В окне Alignment (Выравнивание) выберите в панели Vertical (Вертикальное) вариант Centers (По центру) и нажмите кнопку ОК. Все компоненты, участвующие в операции выравнивания, будут выровнены по вертикали, как показано на рис. 12.18.

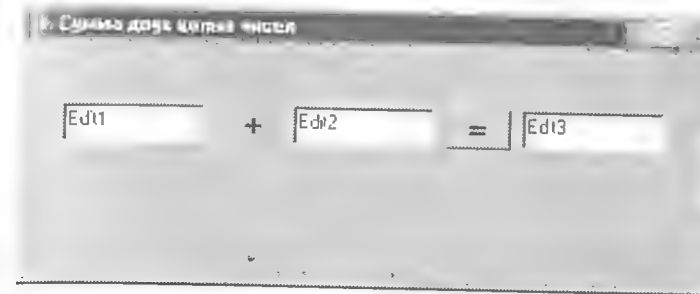


Рис. 12.18. Результат выравнивания компонентов на форме

Удалите текст Edit1, Edit2, Edit3 в соответствующих компонентах. Для этого выберите объект в окне Инспектора объектов, на странице Properties выберите свойство Text и удалите текст, как показано на рис. 12.19.

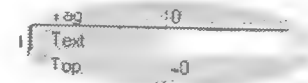


Рис. 12.19. Удаление текста в Edit1.Text

ПРИМЕЧАНИЕ

При работе с Инспектором объектов можно получить контекстную справку по свойствам или событиям. Для этого следует выделить в окне Инспектора объектов интересное свойство или событие и нажать клавишу F1, а затем просмотреть в окне Delphi Help справочную информацию.

Добавьте на форму три объекта Label, расположите их над объектами Edit1-Edit3 и задайте их свойствам Caption значения «Слагаемое», «Слагаемое» и «Сумма». Нажатием F12 активизируйте окно Редактора кода. Обратите внимание, что в разделе описания программного модуля ИСР были сгенерированы описание формы и размещенных на ней компонентов.

```

pe
 TForm1 = class(TForm)
   Edit1: TEdit;
   Edit2: TEdit;
   Edit3: TEdit;
   Label1: TLabel;
   Button1: TButton;
   Label2: TLabel;
   Label3: TLabel;
   Label4: TLabel;

```


СОВЕТ

После того как завершено тщательное размещение и выравнивание компонентов, их местоположение на форме можно зафиксировать, чтобы в дальнейшем случайно не сдвинуть какой-либо компонент. Для фиксации положения компонентов следует воспользоваться командой Edit ► Lock Controls. После этого блокируются даже вновь размещаемые на форме компоненты. Если в дальнейшем возникнет необходимость в изменении расположения компонентов, то нужно будет повторно выбрать команду Edit ► Lock Controls.

Сохраните изменения, внесенные в проект, нажатием кнопки Save All в стандартной панели инструментов. Откомпилируйте созданный проект командой Project ► Compile summa (слово «summa» указывает имя проекта).

Для просмотра содержимого проекта Delphi откройте окно Проводника и просмотрите папку проекта.

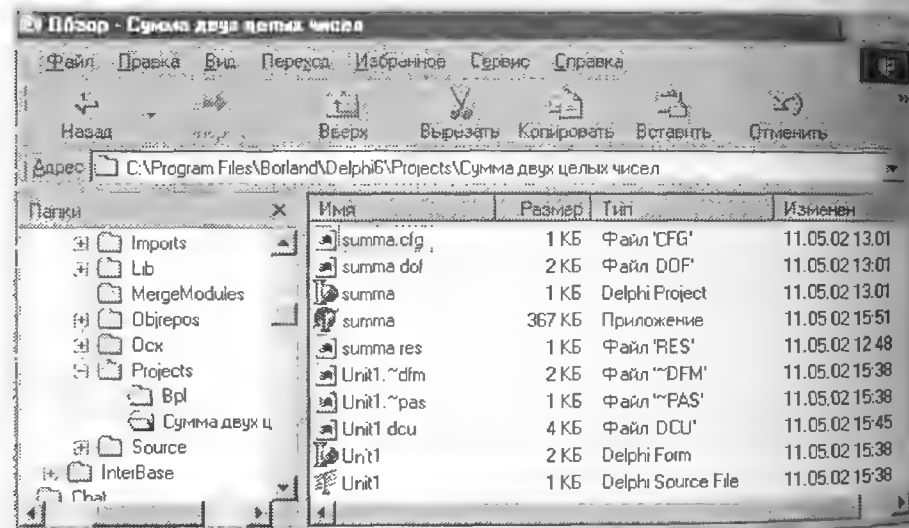


Рис. 12.20. Состав проекта в Delphi

Как показано на рис. 12.20, проект Delphi состоит из форм, модулей, установок параметров проекта, ресурсов и т. д. Вся эта информация размещается в файлах. Многие из этих файлов автоматически создаются Delphi, когда вы строите приложение. Главной частью приложения является файл summa.dpr — файл проекта, содержащий код на языке Object Pascal, с которого начинается выполнение программы и который обеспечивает инициализацию других модулей. Он создается и модифицируется Delphi автоматически в процессе разработки приложения. Имя, присваиваемое файлу проекта при его сохранении, становится именем исполняемого файла (в нашем примере — summa.exe). Другие файлы проекта:

Unit1.pas — файл модуля;

Unit1.dfm — файл формы;

summa.dof — файл параметров проекта;

summa.res — файл ресурсов;

Unit1.~dfm, Unit1.~pas — файлы резервных копий.

ПРИМЕЧАНИЕ

Обратите внимание, что файл модуля Unit1.pas был сгенерирован ИСР Delphi. Пока вам не пришлось вводить ни одного символа в текст программы на языке Object Pascal.

ПРИМЕЧАНИЯ

Список расширений файлов, создаваемых Delphi, выглядит следующим образом:

DPR — файл проекта Delphi. Это главная программа приложения.

~DP — резервный файл DPR. Создается, если включено резервное сохранение.

PAS — исходный код модуля или формы. Главная программа находится в файле DPR.

~PA — резервный файл PAS. Создается, если включено резервное сохранение.

DFM — эти файлы связаны с файлами PAS. DFM — это двоичные файлы, в которых содержатся начальные данные для компонент (это те свойства, которые программист устанавливает в Object Inspector). Файл DFM нельзя отредактировать текстовым редактором, но если открыть его в Delphi, то будет отображена текстовая версия его содержимого.

~DF — резервный файл DFM. Создается, если включено резервное сохранение.

DCU — откомпилированный модуль, подобен файлу OBJ.

DSM — файл с информацией, используемой внутренним отладчиком. Этот файл создается при каждой компиляции проекта.

OPT — опции проекта. Это настройки компилятора и компоновщика для данного проекта. Их можно вызвать в меню Options/Project.

RES — файл ресурсов Windows. Создается Delphi автоматически и требуется для компиляции. Пользоваться этим файлом необязательно, но удалять его не следует.

EXE — откомпилированный проект. Исполняемая Windows-программа.

Кроме того, в папке проекта могут находиться и другие файлы Windows, которые используются в проекте Delphi:

Файлы справки (.hlp);

Файлы изображений или графические файлы (.wmf, .bmp, .ico).

После просмотра списка файлов проекта закройте окно Проводника и активизируйте окно Delphi.

Запустите программу на выполнение при помощи команды Run ► Run или нажатием кнопки Run в панели инструментов или клавиши F9. После этого на экране компьютера появится окно созданной вами формы с компонентами, в которые можно вводить значения. Обратите внимание, что окно формы имеет стандартные атрибуты окна Windows, его можно минимизировать, развернуть на весь экран, изменить его размеры, но приложение не будет выполнять каких-либо вычислений, так как выполняющий вычисления фрагмент программы не был создан. Завершите работу приложения любым из стандартных способов, например, нажатием Alt+F4.

Создание кода — обработчика события

Для того чтобы приложение выполняло вычисления при щелчке на кнопке Button1 с изображением символа «=», следует написать код обработки этого события. Для создания кода обработки события следует воспользоваться Инспектором объектов. В окне Инспектора объектов на странице Events указаны все события, на которые может реагировать выбранный объект. Страница событий связана с Редактором кода следующим образом: если дважды щелкнуть мышью справа от какого-либо пункта, то соответствующий данному событию код будет автоматически помещен в окно Редактора кода, а окну Редактора кода немедленно получит фокус, и вы сразу же будете иметь возможность отредактировать код обработчика данного события. Например, если требуется выполнить определенные действия при щелчке левой кнопкой мыши по данному объекту, то следует выделить событие OnClick. Рядом с именем этого события откроется окно с раскрывающимся списком. Если в приложении уже были созданы какие-нибудь обработчики событий и при событии OnClick требуется использовать один из них, можно выбрать необходимый обработчик из раскрывающегося списка.

ПРИМЕЧАНИЕ

В Delphi 5-6 в Инспектор объектов введена возможность фильтрации свойств и событий и возможность группировать их по категориям. Чтобы воспользоваться этими возможностями, следует щелкнуть в окне Инспектора объектов правой кнопкой мыши. Если выбрать в раскрывающемся меню раздел View, то будет показан список категорий свойств и событий. Около каждой категории имеется индикатор. Можно включить индикаторы только у некоторых категорий, и тогда в Инспекторе объектов будут отображаться события и свойства только указанных категорий. Выбор раздела Toggle переключает видимость разделов: видимые становятся невидимыми, и наоборот. Выбор раздела All позволяет сделать видимыми все свойства и события, а выбор раздела None позволяет сделать все события и свойства невидимыми. В нижней части окна Инспектора объектов указывается, сколько свойств или событий невидимы в данный момент.

Так как в приложении еще нет обработчика событий, требуется написать новый обработчик щелчка на кнопке Button1 с изображением символа «=». Для этого следует выбрать в окне Инспектора объектов объект Button1, затем на странице Событий произвести двойной щелчок на пустом поле списка в событии OnClick. После этого окно Редактора кода немедленно получит фокус. В этом окне в разделе interface находится запись процедуры обработчика события:

```
procedure TForm1.Button1Click(Sender: TObject);
```

В разделе implementation располагается текст заготовки этой процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ...
end;
```

Курсор будет находиться в пустой строке между ключевыми словами begin и end. Этот код является заготовкой для обработчика события, которую автоматически создала ИСР Delphi. Вам остается только разместить необходимые операторы в пространстве между begin и end. Так как в задаче требуется выполнить сложение двух целых чисел, а результат их сложения также является целым числом, то в разделе описания переменных следует ввести следующее описание:

```
var
    a,b,c: integer;    {2 слагаемых и сумма - целые числа}
```

Так как в приложении для ввода чисел-слагаемых используются окна редактирования Edit1, Edit2, то необходимо при помощи процедуры StrToInt преобразовать строки из окон Edit1, Edit2 в целые числа. Для вывода результата суммирования в окне редактирования Edit3 нужно преобразовать число в строку функцией IntToStr. Поэтому в основное тело процедуры обработки события следует ввести следующий текст:

```
a:=StrToInt(Edit1.text);
b:=StrToInt(Edit2.text);
c:=a+b;
Edit3.text:=IntToStr(c);
```

Целиком процедура обработки события щелчка на кнопке Button1 будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    a,b,c: integer;    {2 слагаемых и сумма - целые числа}
begin
    a:=StrToInt(Edit1.text); {преобразование текстовой строки в целое число}
    b:=StrToInt(Edit2.text);
    c:=a+b; {вычисление суммы}
    Edit3.text:=IntToStr(c); {преобразование целого числа в текстовую строку}
end;
```

ПРИМЕЧАНИЕ

В нижней части окна Редактора кода располагается строка состояния. В самой левой ее позиции отображается положение курсора: номер строки и колонки. Второй элемент строки — индикатор модификации. Если код был изменен с момента последнего сохранения его в файле, то в индикаторе модификации появляется слово «Modified», означающее, что код был изменен и изменения не сохранены на диске. Третий элемент строки состояния — индикатор режима вставки, показывающий, будут ли вводимые символы вставляться в текст или заменять его. Переключение режима Вставка/Замена производится клавишей Insert.

В окне Редактора кода, как и в другие окна Delphi, встроена контекстная справка. Чтобы получить справку по какому-либо слову кода (ключевому слову, имени функции и т. п.), например, StrToInt, следует установить курсор на это слово и нажать клавишу F1. После этого откроется окно Delphi Help со справочной информацией, как показано на рис. 12.21.

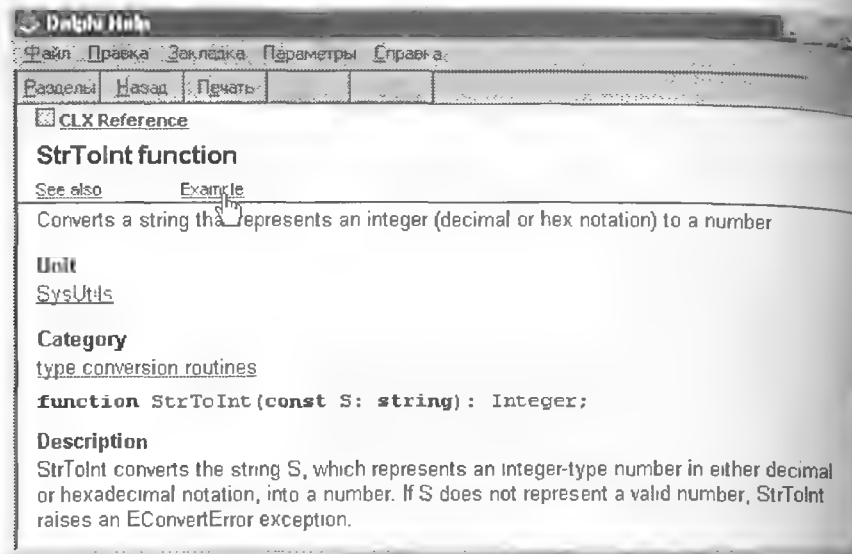


Рис. 12.21. Справка Delphi по функции StrToInt

Закройте окно справки и сохраните изменения в проекте, нажав на кнопку Save All в панели инструментов ИСР.

Запустите приложение на выполнение нажатием кнопки Run в панели инструментов. В окне приложения введите значения слагаемых и, нажав кнопку Button1 с надписью «=», запустите процедуру обработки события — вычисление суммы. Окно созданного приложения будет выглядеть следующим образом:

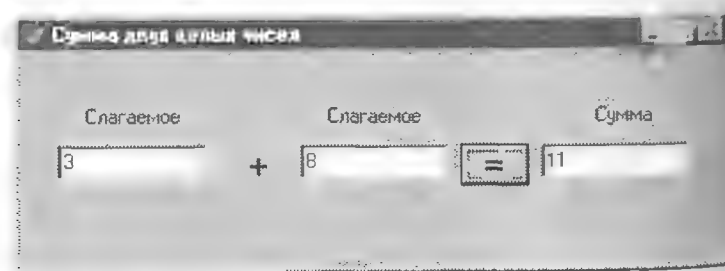


Рис. 12.22. Вид окна приложения

После проверки работы приложения закройте его окно.

Открытие ранее сохраненного проекта

Чтобы открыть сохраненный ранее проект, выберите в меню File команду Open Project. В окне Open Project выберите папку, файл проекта и нажмите кнопку Открыть. Можно открыть проект из списка редактированных в последнее время, выбрав команду File ► Recent, а затем в списке последних редактировавшихся проектов выбрать нужный, например, summa.

Изменение свойств визуальных компонентов

Упражнение 2. В проекте summa измените свойства компонентов Label2, Label3, Label4, задав для этих компонентов стиль шрифта «курсив», цвет шрифта «синий», размер символов «10 пунктов».

Для изменения свойств компонента Label2 в окне Инспектора объектов выберите компонент Label2. На странице Properties разверните группу свойств Font двойным щелчком на свойстве Font. Откроется таблица свойств шрифта, содержащая свойства Color (цвет), Name (имя шрифта), Size (размер символа) и др. Щелкнув на свойстве Color, выберите в списке цвет clBlue (синий), как показано на рис. 12.23.

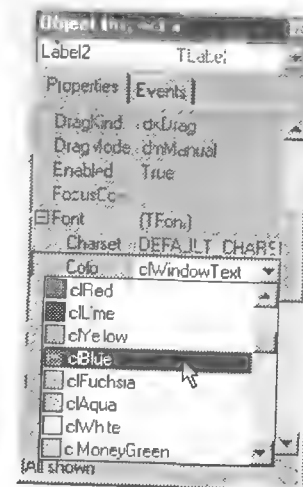


Рис. 12.23. Определение свойства Font.Color

ПРИМЕЧАНИЕ

Двойной щелчок в поле значений свойства Color открывает окно Цвет для выбора цвета символов. В этом окне можно выбрать или, щелкнув на кнопке Определить цвет, определить требуемый цвет в палитре, расположенной в правой части окна Цвет, показанной на рис. 12.24, и нажать кнопку ОК. Шестнадцатеричный код выбранного цвета, например, \$00B8E61A, будет записан в поле значений свойства Font.Color.

Щелчком мышью в окне Инспектора объектов на свойстве Size активизируйте поле ввода значений и задайте размер символов 10 пунктов. Дважды щелкнув на свойстве Style (стиль), раскройте дополнительный список подвнимание, в котором установите в true свойство fsItalic (курсив). Обратите внимание, что у надписи Label2 на форме изменились цвет, размер символов и стиль начертания. Аналогичным образом измените цвет, размер символов и стиль начертания надписей Label3 и Label4.

Упражнение 3. Создайте приложение, реализующее ввод двух целых чисел, по щелчку на кнопке с символом «=» вычисляющее результат операции вещественного деления и выводящее значение результата на экран.

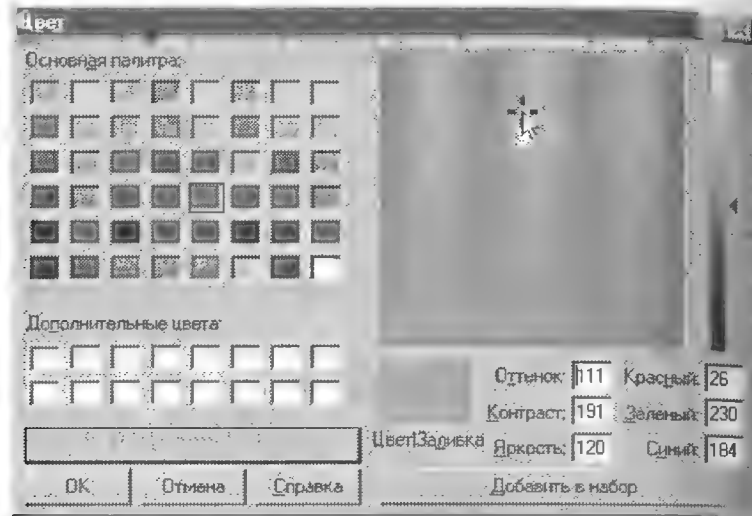


Рис. 12.24. Окно выбора цвета

Запустите ИСР Delphi и создайте новый проект командой **File** ► **New** ► **Application**. В результате создания проекта в окне Delphi будет раскрыто окно формы, на которой вы будете размещать визуальные компоненты проекта. Сохраните новый проект при помощи команды **File** ► **Save Project As**. В окне **Save As** создайте новую папку для файлов создаваемого проекта, например, «Вещественное деление». Затем в окне **Save As** откройте созданную папку, задайте в поле **Имя файла** имя **main** и щелкните на кнопке **Сохранить**. После сохранения файла модуля **main.pas** откроется окно **Save Project As**, в котором следует задать имя файла проекта, например, **Delenie**, а затем нажать кнопку **Сохранить**.

Измените свойства формы. Для изменения размеров формы захватите угол окна формы и, не отпуская левую кнопку мыши, перемещайте мышью, задавая требуемый размер формы. Измените надпись в заголовке формы **Form1** с помощью Инспектора объектов. Для этого в окне Инспектора объектов выберите объект **Form1**, откройте страницу **Properties** (Свойства), затем выберите свойство **Caption** (Заголовок) и задайте новое значение этого свойства, например, «Вещественное деление двух целых чисел».

Разместите на форме компоненты **Edit1**, **Edit2**, **Edit3**, **Label1**, **Label2**, **Label3**, **Label4** и **Button1**, как показано на рис. 12.25.

Удалите текст **Edit1**, **Edit2**, **Edit3** из соответствующих компонентов, для чего выберите объект в окне Инспектора объектов, на странице **Properties** выберите свойство **Text** и удалите текст, как показано на рис. 12.19.

Задайте для свойства **Caption** компонента **Label1** значение «/». Измените размер символов компонента **Label1**, для чего в окне Инспектора объектов выберите в списке объект **Label1**, затем на странице **Properties** разверните список свойств **Font**. В списке свойств **Font** выберите свойство **Size** и задайте для него значение 20, как показано на рис. 12.16. Аналогичным образом задайте для свойства **Caption** компонента **Button1** значение «=» и размер символов 20. Задайте для свойства **Caption** компонента **Label2** значение «Делимое», для **Label3** — «Делитель», для **Label4** — «Частное». Зафиксируйте положение компонентов на форме, выбрав команду **Edit** ► **Lock Controls**. После редактирования свойств визуальных компонентов форма приложения будет выглядеть следующим образом:

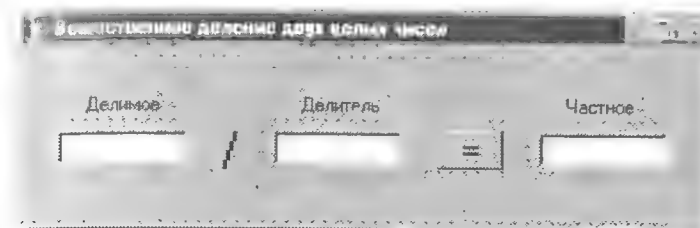


Рис. 12.25. Вид формы приложения с визуальными компонентами

Нажав **F12**, откройте окно Редактора кода и просмотрите текст модуля, сгенерированный Delphi в процессе визуального проектирования формы. Обратите внимание, что в тексте программы еще не описаны переменные и нет процедуры вычисления операции вещественного деления.

Чтобы приложение выполняло вычисления при щелчке мышью на кнопке **Button1** с изображением символа «=», следует написать код обработки этого события. Для создания кода обработчика события следует воспользоваться помощью Инспектора объектов. Выберите в его окне объект **Button1**, затем на странице События дважды щелкните мышью на пустом поле списка в событии **OnClick**. После этого окно Редактора кода немедленно получит фокус, в котором в разделе **interface** появится запись процедуры обработчика события:

```
procedure TForm1.Button1Click(Sender: TObject);
```

В разделе **implementation** появится текст заготовки этой процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
end;
```

```
var
```

Курсор будет расположен в пустой строке между ключевыми словами **begin** и **end**. Этот код является заготовкой обработчика события, которое будет выполняться при щелчке мышью на кнопке **Button1**. Остается только разместить в промежутке между **begin** и **end** необходимые операторы. Так как в задаче необходимо выполнить деление двух целых чисел, а результат их вещественного деления всегда будет вещественным числом, то в разделе описания переменных следует ввести следующее описание:

```
var
  a,b : integer; {2 операнда - целые числа}
  c : real;      {частное - вещественное число}
```

В тело процедуры обработки события введите текст:

```
a:=StrToInt(Edit1.text); {преобразование текстовой строки в целое
                           число}
b:=StrToInt(Edit2.text);
c:=a / b;                {выполнение операции вещественного деления
                           и присваивание результата переменной c}
Edit3.text:=FloatToStrF(c,ffGeneral,7,4);
                           {преобразование вещественного числа
                           в текстовую строку}
```

ПРИМЕЧАНИЕ

Поскольку свойства text объектов Edit1, Edit2 имеют строковые значения, для их преобразования в целые числа используется стандартная функция Object Pascal — StrToInt. Так как свойству Edit3.text нужно присвоить значение вещественного частного, то для преобразования вещественного числа в текстовую строку используется функция FloatToStrF.

Для получения справочной информации о синтаксисе функции FloatToStrF укажите курсором название функции и нажмите F1. В окне справки описано назначение параметров этой функции. Откомпилируйте и запустите программу на выполнение при помощи команды Run ► Run или щелчка мышью на кнопке Run в панели инструментов или нажатием клавиши F9. После этого на экране компьютера появится окно созданной вами формы с компонентами Edit1, Edit2, в которые можно вводить значения, как показано на рис. 12.26.

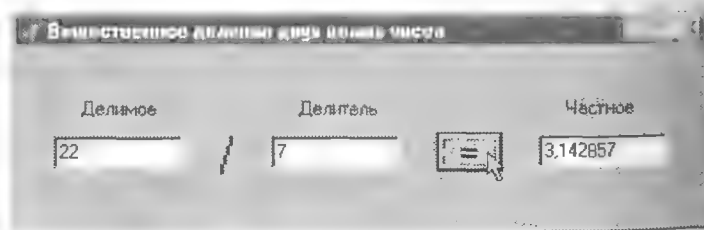


Рис. 12.26. Окно приложения, выполняющего операцию вещественного деления

При щелчке на кнопке Button1 «=>» обработчик этого события будет выполнять операцию вещественного деления двух чисел и выводить частное в окне Edit3. Завершите работу приложения любым из стандартных способов, например, щелчком мышью на кнопке Закрывать в правом верхнем углу окна.

Упражнение 4. Дополните программу обработчика события таким образом, чтобы перед операцией деления выполнялась проверка делителя на равенство нулю. В случае равенства делителя нулю вместо выполнения деления в окне Edit3 должно отображаться сообщение «На ноль делить нельзя!».

Вставьте в процедуру обработчика события TForm1.Button1Click операцию if then else, а в качестве условия выполнения операции вещественного деления задайте $b \neq 0$. Фрагмент программы, проверяющий условие отличия

делителя от нуля и выполняющий деление, если условие выполняется, может быть записан следующим образом:

```
if b<>0 then {проверка отличия делителя от 0}
begin
  c:=a / b; {выполнение операции вещественного деления и присваивание
             результата переменной c}
  Edit3.text:=FloatToStrF(c,ffGeneral,7,4);
             {преобразование вещественного числа в текстовую строку}
end
else {вывод сообщения}
  Edit3.text:='На ноль делить нельзя!'
```

Сохраните текст модуля под именем main1.pas, воспользовавшись командой File ► Save As, затем в окне Save As задайте в поле Имя файла имя main1 и щелкните мышью на кнопке Сохранить. Сохраните измененный проект под именем delenie1 в той же папке при помощи команды File ► Save Project As, затем в окне Save As задайте в поле Имя файла имя delenie1 и щелкните мышью на кнопке Сохранить.

При помощи команды View ► Units или клавиш Ctrl+F12 в окне View Unit выберите программу delenie1 и отредактируйте ее текст, изменив описание раздела uses следующим образом:

```
uses
  Forms,
  main1 in 'main1.pas' {Form1};
```

Откомпилируйте модифицированную программу командой Project ► Compile. Проверьте работу приложения, задавая в качестве делителя разные значения.

Упражнение 5. Выполните отладку приложения в пошаговом режиме. Для этого командой View ► Debug Windows ► Watches откройте окно просмотра значений переменных и выражений. Для включения переменной или выражения в окно просмотра выделите в окне Редактора кода переменную или выражение и нажмите Ctrl+F5.

ПРИМЕЧАНИЕ

Среда разработки Delphi 6 поддерживает операции перетаскивания во время отладки. Например, из редактора кода можно перенести выражение в окно Watch List, после чего это выражение останется в соответствующем списке.

Для копирования выражения $b \neq 0$ из окна Редактора кода в окно Watch List выделите в тексте модуля main1 выражение $b \neq 0$ и переместите его мышью в окно Watch List. Список переменных и выражений для просмотра в окне может выглядеть, как показано на рис. 12.27.

Выполните приложение в пошаговом режиме при помощи последовательных нажатий клавиши F7, наблюдая в окне Watch List за изменением значений

переменных и выражений. Обратите внимание, что в случае ввода нуля в качестве делителя значение выражения $b \neq 0$ в окне Watch List становится равным False, а Edit3.text присваивается значение: «На ноль делить нельзя!».

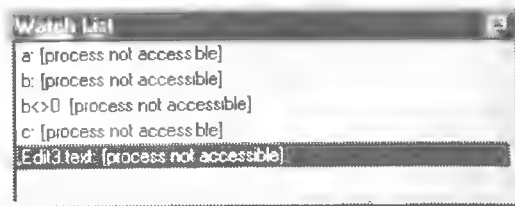


Рис. 12.27. Список переменных и выражений в окне Watch List

Упражнение 6. Чтобы сообщение «На ноль делить нельзя!» выводилось красным цветом, дополните код следующим образом:

```
begin
    Edit3.Font.Color:=clRed;    {установить красный цвет шрифта}
    Edit3.text:='На ноль делить нельзя!'
end;
```

Откомпилируйте и запустите на выполнение модифицированную программу. Проверьте ее работу, задавая в качестве делителя различные значения. Обратите внимание, что в случае ввода в качестве делителя нуля, сообщение «На ноль делить нельзя!» в окне Edit3 выводится красным цветом. Причем если цвет шрифта был изменен, то он так и остается красным, даже если при следующей итерации делитель не будет равен нулю. Чтобы восстановить черный цвет шрифта в окне Edit3, дополните код модуля main1 перед оператором if then else следующей строкой:

```
Edit3.Font.Color:=clBlack;    {установить черный цвет шрифта}
```

Откомпилируйте и запустите приложение на выполнение. Проверьте его работу, задавая в качестве делителя разные значения. Убедитесь, что процедура обработчика события нажатия на кнопку Button1 правильно обрабатывает все варианты значений делителя и изменяет цвет шрифта в окне Edit3. Обратите внимание, что текст сообщения «На ноль делить нельзя!» отображается в окне Edit3 не целиком. Закройте окно приложения.

Упражнение 7. Измените программу таким образом, чтобы ширина окна Edit3 изменялась в ходе выполнения программы и при выводе сообщения «На ноль делить нельзя!» ширина окна Edit3 обеспечивала бы отображение всего текста сообщения.

Переключитесь в окно Инспектора объектов, выберите в списке объектов Edit3 и проверьте значение свойства Width (ширина). В нашем примере оно равно 81 пикселу. Для того чтобы ширина окна Edit3 изменялась в ходе выполнения программы и обеспечивала отображение всего текста сообщения

«На ноль делить нельзя!», отредактируйте код модуля main1, вставив перед оператором вывода сообщения Edit3.text:='На ноль делить нельзя!' строку:

```
Edit3.Width:=130;    {увеличение ширины Edit3}
```

Чтобы восстановить значение ширины окна Edit3, заданное при проектировании формы, введите перед оператором if then else следующую строку:

```
Edit3.Width:=81;    {восстановить первоначальную ширину Edit3}
```

Откомпилируйте и запустите на выполнение отредактированный проект. Проверьте работу программы, задавая в качестве делителя разные значения. Убедитесь, что процедура обработчика события нажатия на кнопку Button1 правильно обрабатывает все варианты значений делителя и изменяет цвет шрифта и ширину окна Edit3 при выводе сообщения «На ноль делить нельзя!», как показано на рис. 12.28.

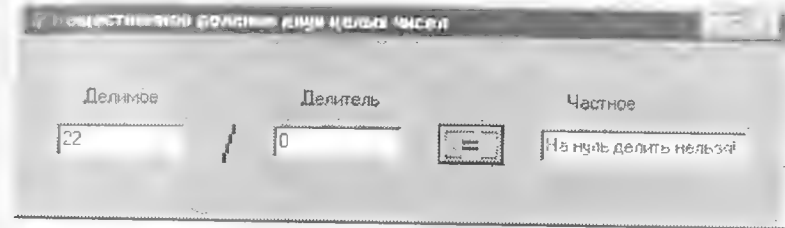


Рис. 12.28. Вид окна приложения с изменяющимся по ширине окном Edit3

Обратите внимание, что в случае корректного ввода делителя, не равного нулю, ширина окна Edit3 уменьшается до значения, заданного при проектировании формы. Закройте окно приложения и сохраните внесенные в проект изменения, щелкнув мышью на кнопке Save All в панели инструментов Delphi.

Упражнение 8. Измените программу таким образом, чтобы текст «На ноль делить нельзя!» выводился в отдельном окне сообщения.

ПРИМЕЧАНИЕ

Для вывода текста в отдельном окне сообщения используйте вызов процедуры ShowMessage(const Msg: string), которая отображает модальное окно сообщения с кнопкой OK. Текст сообщения задайте параметром Msg, в нашем примере — 'На ноль делить нельзя!'.

Чтобы не вносить изменения в проект, созданный в предыдущих заданиях, сохраните проект под новым именем delenie2, а текст программы — под именем main2.

Отредактируйте текст тела модуля main2 следующим образом:

```
begin
    a:=StrToInt(Edit1.text);    {преобразование текстовой строки
                                в целое число}
    b:=StrToInt(Edit2.text);
```



```

Edit3.text:=''
if b=0 then ShowMessage('На ноль делить нельзя!')
                                {вывод текста в окне сообщения}
else
                                {делитель отличен от нуля}
begin
  c:=a/b;                       {выполнение вещественного деления}
  Edit3.text:=FloatToStr(c);
                                {преобразование вещественного числа
                                в текстовую строку}
end;
end;

```

Выберите команду **View ► Units**, в окне **View Unit** выберите программу **delenie2** и отредактируйте ее текст, изменив описание раздела **uses** следующим образом:

```

uses
  Forms,
  main2 in 'main2.pas' {Form1};

```

Сохраните проект. Откомпилируйте и запустите приложение на выполнение. Проверьте его работу, задавая в качестве делителя разные значения. Обратите внимание, что в случае ввода в качестве делителя нуля в модальном окне сообщения выводится сообщение «На ноль делить нельзя!», как показано на рис. 12.29.

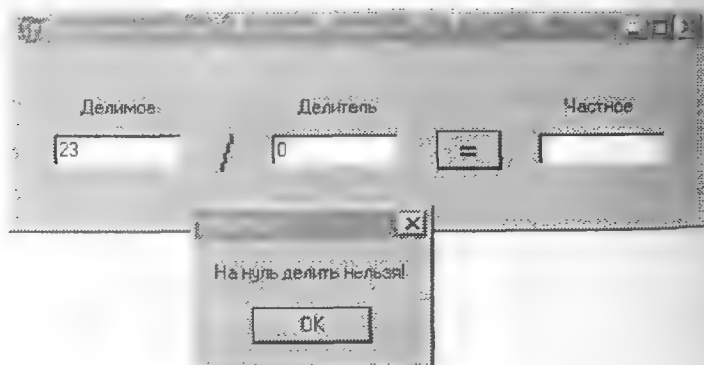


Рис. 12.29. Вывод сообщения в модальном окне

Для продолжения работы с приложением необходимо закрыть окно сообщения, щелкнув мышью на кнопке **OK**.

Упражнение 9. Измените значок приложения **Delenie2**. Для вызова редактора изображений выберите в меню **Delphi** команду **Tools ► Image Editor**. Изучите окно **Image Editor**. Как видно на рис. 12.30, оно очень напоминает окно стандартного графического редактора **Paint**. В окне редактора изображений командой **File ► Open** откройте файл ресурсов приложения. В диалоговом окне **Открытие файла** выберите папку с проектом (в нашем примере — Вещественное деление), в поле **Тип файла** выберите вариант **Resource/DCR files (*.res;*.dcr)**.

в списке файлов выберите файл ресурсов приложения (в нашем примере — **Delenie2.res**) и нажмите кнопку **Открыть**.

Как видно на рис. 12.30, в окне структуры файла ресурсов имеется стандартный узел **Icon**. Щелкнув мышью по символу **+** в строке **Icon**, разверните узел. Щелкнув мышью по вершине **MAINICON**, соответствующей стандартному значку приложения, откройте значок для редактирования (рис. 12.31).

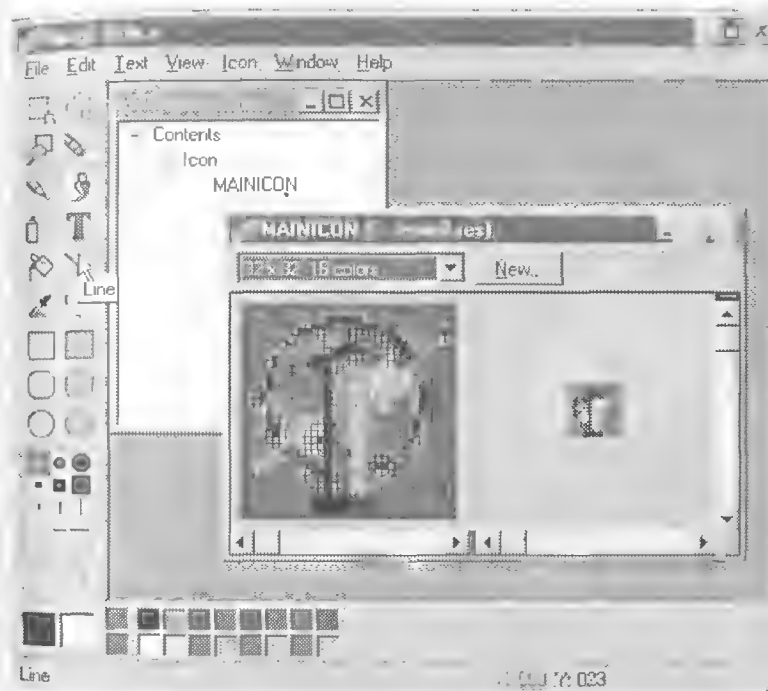


Рис. 12.30. Редактирование значка приложения в Image Editor

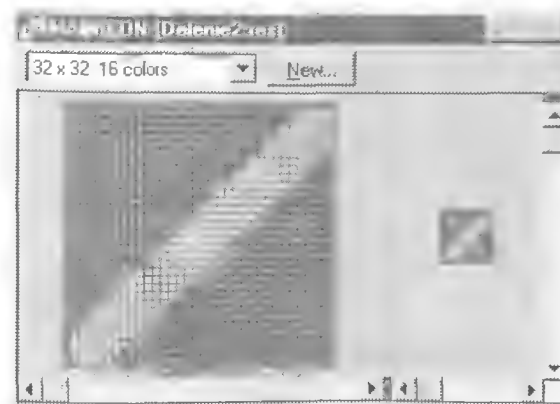


Рис. 12.31. Отредактированный значок

Закройте окно редактирования значка командой **File** ► **Close**. Сохраните изменения в файле ресурсов командой **File** ► **Save**. Закройте окно редактора **Image Editor** командой **File** ► **Exit**.

В ИСР Delphi откройте проект приложения **Delenie2** при помощи команды **File** ► **Open Project**, в окне **Open Project** откройте нужную папку и, выбрав в ней файл проекта приложения, например, **Delenie2**, нажмите кнопку **Открыть**. Откомпилируйте и запустите приложение на выполнение. Обратите внимание, что значок приложения изменился, как показано на рис. 12.32.

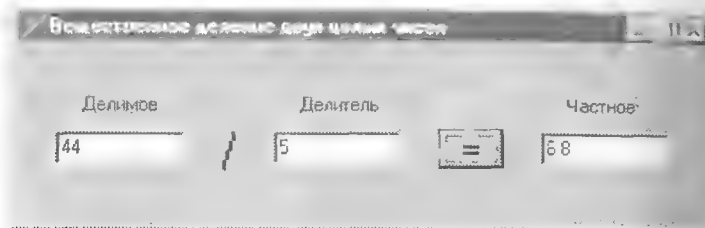


Рис. 12.32. Окно приложения с измененным значком

Упражнение 10. Создайте приложение, обеспечивающее ввод двух целых чисел и выполнение над ними арифметических операций сложения, вычитания, умножения и вещественного деления. Для выбора операции используйте переключатели, вывод сообщения об ошибке при вводе делителя, равному нулю, выполните в отдельном окне сообщений.

После запуска ИСР Delphi создайте новый проект командой **File** ► **New** ► **Application**. Сохраните новый проект командой **File** ► **Save Project As**. В окне **Save Unit1 As** создайте новую папку для файлов создаваемого проекта, например, «Калькулятор». Затем в окне **Save Unit1 As** откройте созданную папку, в поле **Имя файла** задайте имя модуля **Main** и нажмите кнопку **Сохранить**. После сохранения файла модуля **main.pas** откроется окно **Save Project As**. В этом окне в поле **Имя файла** задайте имя файла проекта, например, **Calculator** и нажмите кнопку **Сохранить**.

Задайте для свойства **Form1.Caption** значение «Калькулятор». Разместите на форме **Form1** компоненты **Edit1**, **Edit2** и **Label1**, **Label2**. Задайте для свойства **Caption** компонентов **Label1** и **Label2** значения «Операнд». При нажатой клавише **Shift** выделите компоненты **Edit1**, **Edit2**, **Label1**, **Label2** и, вызвав контекстное меню, командой **Position** ► **Align** ► **Horizontal** ► **Centers** выровняйте их по горизонтали.

Для выбора одной из четырех арифметических операций над операндами используйте переключатели, размещенные на панели **RadioGroup**.

ПРИМЕЧАНИЕ

Компонент **RadioGroup** позволяет отображать и редактировать поля с ограниченным множеством возможных значений.

Разместите на форме **Form1** компонент **RadioGroup1** из палитры **Standard**. Задайте для свойства **Caption** компонента **RadioGroup1** значение «Операция». Так как количество переключателей в группе и надписи около них определяются свойством **Items**, выберите в Инспекторе объектов компонент **RadioGroup1**, а на странице свойств выберите свойство **Items** (список элементов). В окне **String List Editor** введите список элементов — символов арифметических операций: **+**, **-**, *****, **/**, как показано на рис. 12.33.

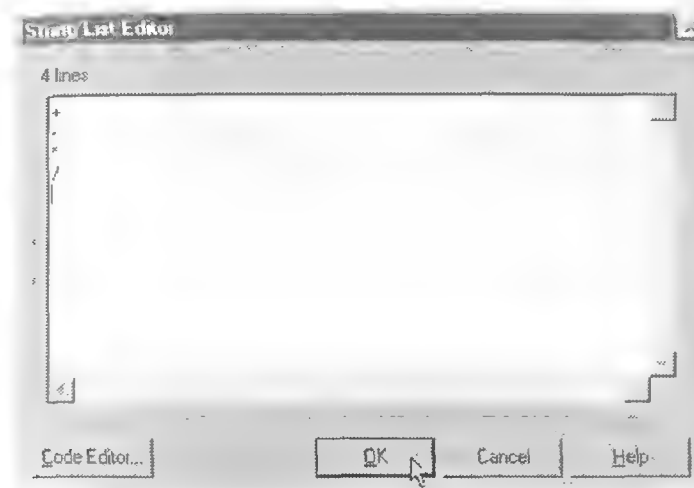


Рис. 12.33. Редактирование списка элементов в панели **RadioGroup**

Щелкнув мышью на кнопке **OK**, завершите формирование списка арифметических операций.

В окне Инспектора объектов задайте для свойства **RadioGroup.ItemIndex** значение 0, чтобы сделать первую кнопку (сложение) выбранной по умолчанию.

ПРИМЕЧАНИЕ

Определить во время выполнения, какая кнопка выбрана в данный момент, можно по индексу **ItemIndex** (0 означает первую кнопку, -1 — ни одна кнопка не выбрана).

Задайте размер символов компонента **RadioGroup**, установив для свойства **RadioGroup.Font.Size** значение 12 пунктов.

Разместите на форме **Form1** кнопку **Button1** и задайте для нее надпись «Вычислить». Разместите на форме **Form1** компоненты **Edit3**, **Label3** и задайте для свойства **Label3.Caption** значение «Результат». Удалите текст **Edit1**, **Edit2**, **Edit3** в соответствующих компонентах. Для этого выберите объект в окне Инспектора объектов, на странице **Properties** выберите свойство **Text** и удалите текст. Выровняйте компоненты **Edit3** и **Label3**, отцентрировав их по горизонтали. Зафиксируйте положение компонентов на форме, выбрав в меню Delphi команду **Edit** ► **Lock Controls**. Форма будет выглядеть, как показано на рис. 12.34.

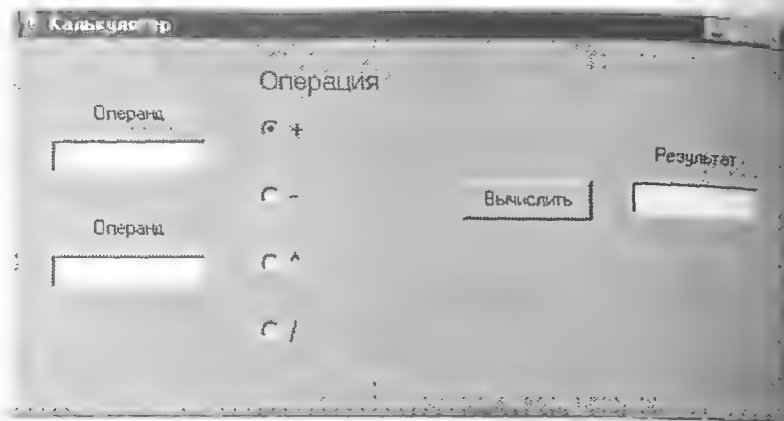


Рис. 12.34. Вид формы приложения с компонентами

Для обработки щелчка на кнопке Button1 с изображением символа «+» в окне Инспектора объектов выберите объект Button1, затем на странице События дважды щелкните мышью на пустом поле списка в событии OnClick. После этого в разделе interface модуля Main.pas появится процедура обработчика события `procedure TForm1.Button1Click(Sender: TObject);`, а в разделе implementation появится текст заготовки этой процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
end;
```

Для решения задачи вычисления арифметических операций над двумя целыми операндами введите в текст процедуры TForm1.Button1Click следующее описание:

```
var
  a, b : integer; { 2 операнда - целые числа }
  c : real; { результат арифметических операций }
```

ПРИМЕЧАНИЕ

Тип переменной `c` — вещественное число — выбран из-за того, что в результате вещественного деления целых чисел результат будет числом вещественного типа.

Так как в процедуре обработки нажатия кнопки Button1 «Вычислить» должно быть 4 варианта реализации, по одному для каждой арифметической операции, то следует записать выбор вычисления с помощью оператора `case`. Для получения подсказки Delphi по синтаксису оператора `case`, введите слово `case`, укажите на него мышью и нажмите F1. В списке найденных разделов выберите раздел **Case statements** и щелкните мышью на кнопке **Показать**. Для просмотра информации используйте полосу прокрутки в окне Delphi Help. Для копирования примера кода программы в окне Delphi Help выделите текст в окне

справки и, нажав правую кнопку мыши, вызовите контекстное меню, затем в контекстном меню выберите команду **Копировать**, как показано на рис. 12.35.

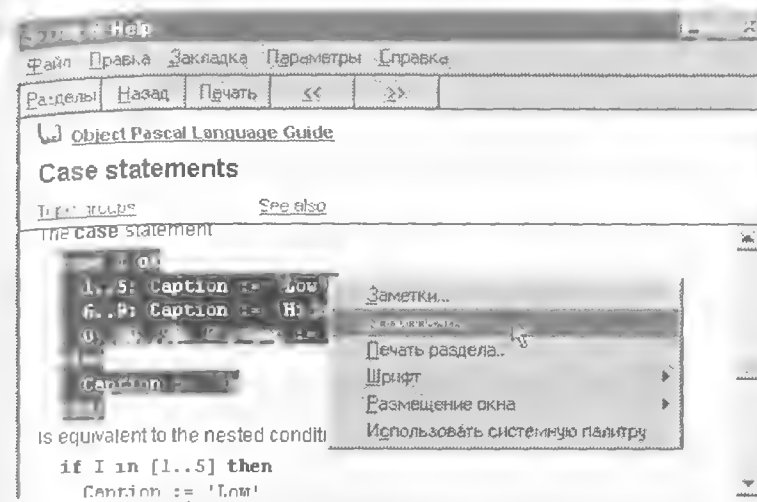


Рис. 12.35. Копирование примера кода программы в окне Delphi Help

В окне Редактора кода укажите место вставки скопированного фрагмента в основном теле процедуры обработки события и, вызвав контекстное меню, выберите команду **Paste (Вставить)**.

В основном теле процедуры обработки события отредактируйте текст оператора `case` и введите текст остальных операторов следующим образом:

```
begin
  a:=StrToInt(Edit1.text); {преобразование текстовой строки
                           в целое число}
  b:=StrToInt(Edit2.text);
  Edit3.text:= ''; {очистить от результата предыдущих вычислений}
                  {выбор операции в зависимости от значения
                  свойства RadioGroup1.ItemIndex}
  case RadioGroup1.ItemIndex of
    0 : c:=a+b; {сложение}
    1 : c:=a-b; {вычитание}
    2 : c:=a*b; {умножение}
    3 : {деление} if b=0 then ShowMessage('На ноль делить нельзя!')
        else c:=a/b;
  end;
  if RadioGroup1.ItemIndex <> 3 then {вывод результата операций}
    Edit3.text:=FloatToStrF(c,ffGeneral,10,4){преобразование результата
                                              вещественного деления в
                                              текстовую строку}
  else if c=0 then
    Edit3.text:=FloatToStrF(c,ffGeneral,10,4)
end;
```

Как видно из текста процедуры, в операторе case выполняется выбор вариантов вычислений, а вывод результата вычислений выполняется оператором, идущим после оператора case, причем для случая $b=0$ результат не выводится и в окне Edit3 не выводится, а выводится сообщение в окне сообщения. Откомпилируйте и запустите приложение на выполнение. На рис. 12.36 показано, как будет выглядеть окно приложения.

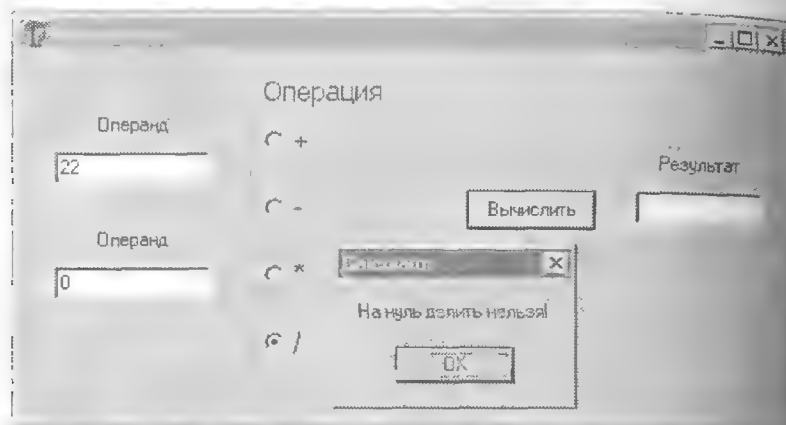


Рис. 12.36. Вид окна приложения с сообщением об ошибке ввода делителя

Проверьте работу приложения для случая, когда не заданы значения операндов. Как видно на рис. 12.37, в этом случае выводится окно сообщения о некорректности значения операнда, закрыть которое можно нажатием кнопки OK.

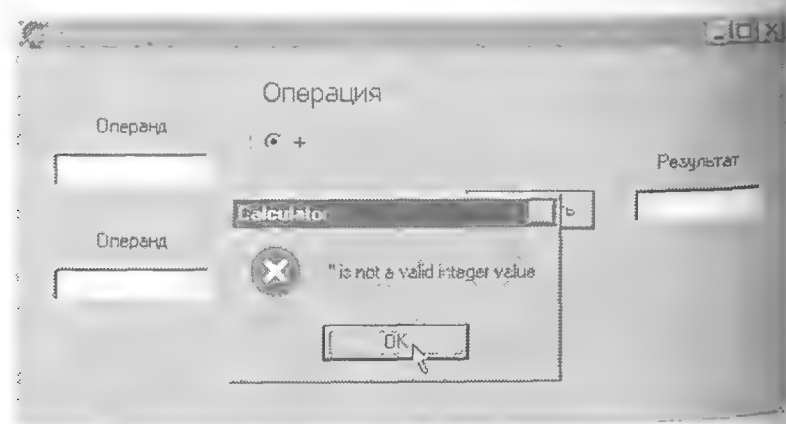


Рис. 12.37. Окно сообщения о некорректности значения операнда

Упражнение 11. Закройте окно приложения и отредактируйте текст модуля таким образом, чтобы перед выполнением вычислений выполнялась проверка, заданы ли значения операндов. Если значения операндов не заданы, то

следует вывести сообщение об этом в отдельном окне. Поместите перед оператором присваивания `a:=StrToInt(Edit1.Text);` строку

```
if (Edit1.Text = '') and (Edit2.Text <> '') then begin
```

которая проверяет значения свойств `Edit1.Text` и `Edit2.Text`. Если эти значения не пустые, то выполняется вычисление арифметической операции, в противном случае управление передается на следующий фрагмент программы, который нужно вставить перед последним оператором `end`;

```
else ShowMessage('Не заданы значения');
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Проверьте работу приложения для случая, когда не заданы значения операндов и убедитесь, что в этом случае выводится сообщение в отдельном окне. Попробуйте ввести в качестве значений операндов не цифры, а символы, например, «а» и «б». Щелкнув на кнопке `Button1`, убедитесь в том, что приложение завершается и выдает сообщение об ошибке, которая возникает из-за того, что приложение пытается преобразовывать символы в число.

Для предупреждения данной ошибки введем обработку события нажатия на клавишу в окне `Edit1` и `Edit2`, чтобы запретить ввод любых символов, кроме цифр от 0 до 9 и знаков `-`, `+`.

Для создания процедуры обработчика события нажатия на клавишу в окне `Edit1` выберите в окне Инспектора объектов компонент `Edit1` и на странице `Events` (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии `OnKeyPress`. После этого окно Редактора кода немедленно получит фокус и в разделе `interface` появится запись процедуры обработчика события

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
```

Параметр `Key` в обработчике этого события соответствует символу нажатой клавиши.

ПРИМЕЧАНИЕ

При этом различаются символы в верхнем и нижнем регистрах и символы кириллицы и латиницы. Клавиши, не имеющие соответствующих им кодов ASCII (`Shift`, `Alt`, `Ctrl` и функциональные клавиши), не вызывают этого события. Поэтому нажатие таких комбинаций клавиш, как, например, `Shift+A`, генерирует только одно событие `OnKeyPress`, при котором параметр `Key` равен «А». Для того чтобы распознавать подобные комбинации клавиш, следует использовать обработчики событий `OnKeyDown` и `OnKeyUp`.

В разделе `implementation` появится текст заготовки этой процедуры:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
```

```
end;
```

Вставьте в тело процедуры следующий оператор:

```
if not (Key in ['0'..'9', '+', '-']) then Key := #0;
```

Действие оператора сводится к сравнению значения переменной `Key` с множеством значений ['0'..'9', '+', '-']. Если символ нажатой клавиши не входит во множество, то `Key` присваивается значение нулевого символа (#0). Таким образом, в окне `Edit1` будет отображаться текст, состоящий из цифр и знаков «+» и «-».

Аналогичным образом создайте процедуру обработчика события нажатия на клавишу в окне `Edit2`. В итоге получится следующий текст модуля приложения:

```
unit main;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    RadioGroup1: TRadioGroup;
    Button1: TButton;
    Edit3: TEdit;
    Label3: TLabel;
  procedure Button1Click(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure Edit2KeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
  {обработчик события щелчка на кнопке Button1}
  var
    a, b: integer; {2 операнда - целые числа}
    c: real;        {результат арифметических операций}
begin
  if (Edit1.Text <> '') and (Edit2.Text <> '') then
  begin
    {если значения заданы}
    a:=StrToInt(Edit1.Text); {преобразование текстовой строки
                              в целое число}
    b:=StrToInt(Edit2.Text);
```

```
    Edit3.Text := ''; {очистить от результата предыдущих
                     {вычислений}
                     {выбор операции в зависимости от
                     значения свойства RadioGroup1.ItemIndex}

  case RadioGroup1.ItemIndex of
    0: c:=a+b: {сложение}
    1: c:=a-b: {вычитание}
    2: c:=a*b: {умножение}
    3: {деление} if b=0 then ShowMessage('На ноль делить нельзя!')
        else c:=a/b;
  end;
  if RadioGroup1.ItemIndex <> 3 then {вывод результата операций}
    Edit3.Text:=FloatToStrF(c, ffGeneral, 10, 4)
    {преобразование результата вещественного
    деления в текстовую строку}

  else if b<>0 then
    Edit3.Text:=FloatToStrF(c, ffGeneral, 10, 4)
  end
  else ShowMessage('Не заданы значения');
end;
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  {разрешить ввод в Edit1 только цифр}
  if not (Key in ['0'..'9', '+', '-']) then Key:=#0;
end;
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  {разрешить ввод в Edit2 только цифр}
  if not (Key in ['0'..'9', '+', '-']) then Key:=#0;
end;
end.
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Попробуйте ввести в качестве значений операндов не цифры, а символы и убедитесь, что созданные выше обработчики событий `OnKeyPress` для `Edit1` и `Edit2` не позволяют пользователю вести нецифровую информацию. Закройте окно приложения.

ПРИМЕЧАНИЕ

При обработке таких исключительных ситуаций Delphi использует специальные объекты — исключения, характеризующие возникшую в программе исключительную ситуацию. В дальнейшем мы познакомимся с обработкой исключений.

Контрольные вопросы и задания

Вопросы

1. Что такое Delphi? Почему пакет получил такое название?
2. Какие компоненты входят в интегрированную среду разработки приложений Delphi?

- Перечислите основные компоненты окна среды Delphi и укажите их значение.
- Как получить подсказку Delphi по элементам окна, компонентам проектируемого графического интерфейса?
- Каково назначение страниц Properties и Events в окне Object Inspector?
- Как разместить компонент на форме?
- Какими способами можно изменять свойства компонента? Приведите примеры из текста.
- Перечислите состав проекта Delphi. Опишите назначение различных файлов.
- Каково назначение обработчиков событий? Каким образом можно инициализировать создание процедуры-обработчика события?
- Опишите порядок получения справки по синтаксису языка Object Pascal в ICP Delphi.
- Как перейти от редактирования кода в модуле к редактированию кода в основной программе?
- Как открыть окно просмотра значений переменных и выражений Watch List? Как включить в это окно переменную или выражение?
- Чем отличается пошаговое выполнение программы в режимах Step Over (F8) и Trace Into (F7)?
- Каково назначение процедуры ShowMessage? Что выступает в качестве параметра процедуры?
- Какое инструментальное средство включено в интегрированную среду Delphi для редактирования файлов ресурсов?
- Опишите назначение компонента RadioGroup. Как задать список элементов-переключателей в панели RadioGroup?
- По какому свойству RadioGroup определяется выбранный переключатель? Как задать этому свойству значение?
- Какие ситуации во время выполнения приложения относятся к исключительным? Каковы способы обработки исключительных ситуаций?

Задания

- Откройте справку Delphi и изучите справочную информацию о компонентах палитры Standard.
- В справке Delphi найдите и изучите информацию о числовых типах данных языка Object Pascal.
- В справке Delphi найдите и изучите информацию об арифметических операциях языка Object Pascal.
- Измените программу вычисления операции вещественного деления двух целых чисел, чтобы она вычисляла результат целочисленного деления и остатка от деления.

- Создайте приложение, которое по заданному натуральному N вычисляло бы числа Фибоначчи, не превышающие заданного N, а также определяло бы количество найденных чисел и их сумму. Вариант вычислений должен определяться по включению соответствующего флажка, как показано на рис. 12.38. Список чисел Фибоначчи следует вывести в компоненте ListBox.

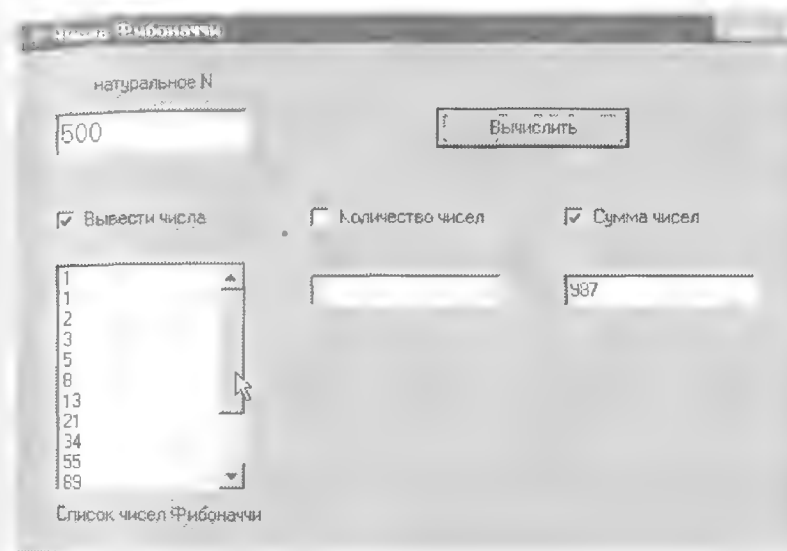


Рис. 12.38. Окно приложения, вычисляющего числа Фибоначчи

ПРИМЕЧАНИЕ

Числа Фибоначчи — это числовая последовательность, где каждый последующий член ряда равен сумме двух предыдущих, т. е.: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 и т. д.

- В справке Delphi найдите и изучите информацию о функции Random в языке Object Pascal.
- В справке Delphi найдите и изучите информацию об операторах повтора While, Repeat, For в языке Object Pascal.
- Создайте приложение «Угадай число», в котором компьютер будет загадывать случайное целое число в диапазоне от 0 до 10, а пользователь должен угадать это число. Ошибка угадывания числа должна сопровождаться сообщением типа «загаданное число меньше» или «загаданное число больше», выводимом в отдельном окне. В случае угадывания пользователем загаданного компьютером числа следует вывести в отдельном окне сообщение «Вы угадали» и указать число попыток.

Глава 13

Приложения для обработки строк, массивов и файлов

В этой главе мы рассмотрим использование интегрированной среды программирования Delphi 6 при создании приложений для обработки структурированных типов данных: строк, массивов и файлов.

Обработка строк типа String

В языке Object Pascal Delphi 5-6 имеется 4 типа строк: String, ShortString, AnsiString, WideString. В справке Object Pascal вы можете изучить эти типы более подробно. В наших примерах рассмотрим использование родового типа String.

Упражнение 1. Создайте приложение, предлагающее пользователю ввести строку, определяющее количество символов в ней и выводящее результат, согласуя окончание слова «символов» с числом символов, например, «В тексте 1 символ», «В тексте 32 символа», «В тексте 47 символов».

СОВЕТ

Для согласования окончания слова с количеством символов используйте оператор case.

Создайте форму и разместите на ней компоненты: Edit1, Label1, Label2, Button1, как показано на рис. 13.1. Задайте значения свойств Label1.Caption — «Введите текст», Label2.Caption — «В тексте», Button1.Caption — «Вычислить». Удалите текст Edit1 из соответствующего компонента. Выровняйте компоненты на форме.

Для вычисления количества символов во введенной строке и вывода результатов создайте процедуру обработчика события щелчка мышью на кнопке Button1. Выберите в окне Инспектора объектов объект Button1, затем на странице Событий произведите двойной щелчок мышью на пустом поле списка в событии OnClick. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.Button1Click(Sender: TObject);.

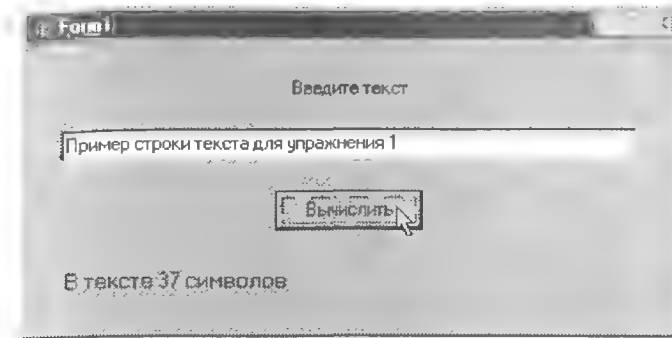


рис. 13.1. Вид окна приложения, подсчитывающего число символов в тексте

Введите в текст процедуры следующее описание переменных:

```
var
  N : integer;      {последняя цифра}
  S : string;       {изменяемое слово "символ"}
```

Для ознакомления со справкой Object Pascal о процедурах и функциях обработки строк введите в окне Редактора кода слово Length и нажмите F1. В списке Найденные разделы выберите тему Length function и нажмите кнопку Показать. В окне справки, прочитав информацию о применении функции Length, щелкните мышью по ссылке Example для просмотра примера. При помощи кнопки Назад в панели инструментов окна Delphi Help вернитесь в предыдущее окно. Для просмотра справочной информации о других процедурах и функциях обработки строк щелкните по ссылке See also (Смотри также). Выбирая в списке Найденные разделы нужную процедуру или функцию, щелкайте мышью на кнопке Показать для просмотра справки. После изучения справочной информации закройте окно справки.

Введите в окне Редактора кода текст тела процедуры обработки текста:

```
begin
  N:=Length(Edit1.Text);           {определить число символов}
  Label2.Caption:='В тексте ' +
  if N>20 then N:=N mod 10;         {выделить последнюю цифру}
  case N of                         {в зависимости от последней цифры}
    1 : S:=' символ';               {изменять окончание слова}
    2..4 : S:=' символа';
    0..5..20 : S:=' символов';
  end;
  Label2.Caption:=Label2.Caption+
  IntToStr(Length(Edit1.Text))+S; {вывести результат}
end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Проверьте работу приложения, задавая текст с разным количеством символов и проверяя правильность вывода результатов работы.

Упражнение 2. Создайте приложение, предлагающее пользователю ввести строку текста, а затем заменяющее символы в тексте и подсчитывающее количество замененных символов. Вариант замены символов должен определяться по положению соответствующего флажка CheckBox, как показано на рис. 13.2. Создайте форму, для свойства Caption задайте значение «Подсчет и замена символов». На форме разместите компоненты: Edit1, Edit2, Edit3, CheckBox1, CheckBox2, CheckBox3, Label1, Label2, Label3, Button1. Задайте значения свойств Label1.Caption — «Исходный текст на русском языке», Label2.Caption — «Измененный текст», Label3.Caption — «Число замен», CheckBox1.Caption — «Подсчитать число замен», CheckBox2.Caption — «Заменить «а» на «б»», CheckBox3.Caption — «Заменить пробелы на тире», Button1.Caption — «Выполнить». Удалите текст Edit1, Edit2, Edit3 из соответствующих компонентов. Выровняйте компоненты на форме, как показано на рис. 13.2.

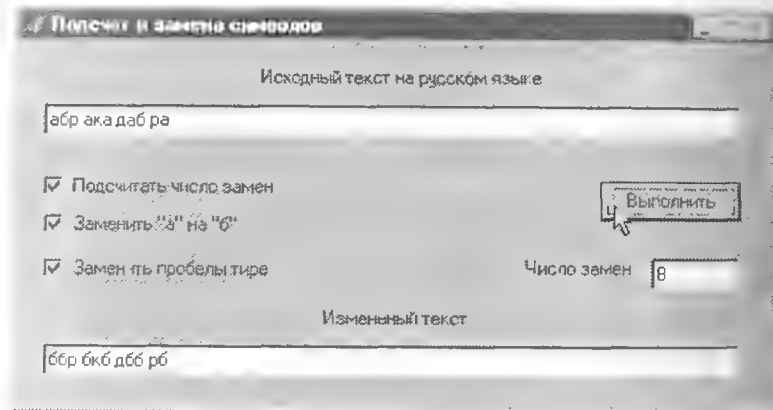


Рис. 13.2. Вид окна приложения, выполняющего замену символов в тексте

Замену символов в тексте и подсчет количества замененных символов опишите в процедуре обработчика события щелчка мышью на кнопке Button1. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект Button1, затем на странице События произведите двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода в заготовку процедуры обработчика события procedure TForm1.Button1Click (Sender: TObject); введите следующее описание переменных:

```
11
var S: string; {строка текста}
    N: byte;    {количество замен символов}
```

СОВЕТ

Для поиска символа в строке используйте функцию Pos языка Object Pascal. Для программирования варианта замены символов используйте значение свойства Checked компонента CheckBox. Если флажок CheckBox1 установлен, то свойство CheckBox1.Checked приобретает значение True.

С учетом вышесказанного тело процедуры может быть записано следующим образом:

```

N:=0 {обнулить число букв а}
Edit3.Text:=''
Edit2.Text:=''
S:=Edit1.Text; {присвоить S значение текстовой строки}
if CheckBox2.Checked then {если включен флажок CheckBox2}
  while Pos('а', S) > 0 do {если в строке найдена буква а}
  begin
    N:=N+1; {увеличить счетчик замен на 1}
    S[Pos('а', S)] := 'б'; {заменить букву а буквой б}
  end;
if CheckBox3.Checked then
  while Pos(' ', S) > 0 do {если в строке найден пробел}
  begin
    N:=N+1; {увеличить счетчик замен на 1}
    S[Pos(' ', S)] := '-'; {заменить пробел символом -}
  end;
if CheckBox1.Checked then Edit3.Text:=IntToStr(N);
                                {вывод числа замен символов}
Edit2.Text:=S; {вывод измененного текста}
end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Проверьте работу приложения, задавая различный текст на русском языке с разным количеством букв «а» и пробелов. Проверьте правильность вывода результатов работы и закройте окно приложения.

Создание и обработка линейного массива

Упражнение 3. Создайте приложение, которое предлагает пользователю задать размер линейного массива, заполняет этот массив случайными целыми числами, выводит список элементов массива, а затем по выбору пользователя определяет минимальный и максимальный элементы массива, сумму всех элементов и количество положительных элементов. Алгоритмы обработки массивов были подробно разобраны при изучении программирования на языке Turbo Pascal.

Создайте форму, для свойства Caption задайте значение «Создание и обработка массива». На форме разместите компоненты Edit1 и Edit2, кнопку Button1, для свойства Caption кнопки задайте значение «Создать массив». Разместите на форме панель GroupBox1, для свойства Caption которой задайте значение «Определить».

ПРИМЕЧАНИЕ

Панель GroupBox — это контейнер с рамкой и надписью, объединяющий группу связанных органов управления, таких как переключатели RadioButton, флажки CheckBox и т. д. GroupBox имеет встроенную рамку с надписью, которая обычно используется для выделения на форме группы функционально объединенных компонентов.

В панели GroupBox1 разместите компоненты CheckBox1, CheckBox2, CheckBox3 и CheckBox4, для свойств Caption которых задайте значения: «Минимальный элемент», «Максимальный элемент», «Сумма всех элементов», «Число положительных элементов», соответственно. Напротив них разместите компоненты Edit3, Edit4, Edit5 и Edit6. Если компоненты CheckBox1, CheckBox2, CheckBox3, CheckBox4 окажутся размещенными под панелью GroupBox1 и не будут отображаться, то следует выделить панель GroupBox1 и выбрать в контекстном меню команду Control ► Send to Back (Порядок ► На задний план). В нижней части формы разместите кнопку Button1 и задайте значение свойства Button1.Caption — «Вычислить». Удалите текст Edit1, Edit2, Edit3, Edit4, Edit5, Edit6 из соответствующих компонентов. Выровняйте компоненты на форме, как показано на рис. 13.3. Зафиксируйте положение компонентов на форме, выбрав в меню Delphi команду Edit ► Lock Controls.

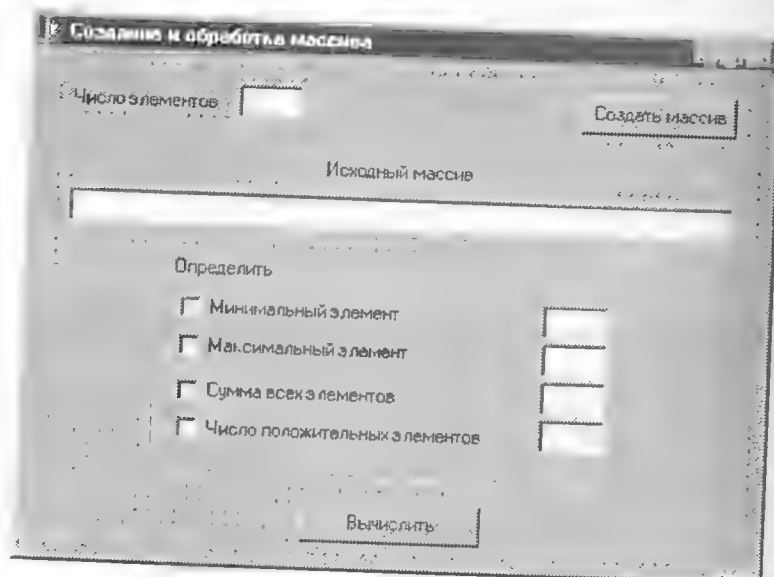


Рис. 13.3. Окно формы с компонентами

Сохраните файл проекта и программного модуля.

Прежде чем создавать обработчики событий щелчка мышью по кнопкам Button1 и Button2, в разделе описания переменных опишите переменные целого типа N и I, где N — размер массива, а I — порядковый номер элемента массива, а также M — динамический массив целых чисел.

```

10 Form1:
11   integer:
12   M: array of integer; {описание динамического массива целых чисел}

```

ПРИМЕЧАНИЕ

Динамические массивы введены в Object Pascal только начиная с версии Delphi 5. Они отличаются от обычных статических массивов тем, что для них не объявляется заранее длина — число элементов. Объявление динамического массива содержит только его имя и тип элементов (один из базовых типов). При объявлении динамического массива место под него не отводится. Прежде чем использовать массив, надо задать его размер процедурой SetLength.

Для предупреждения ввода в окно Edit1 нечислового значения реализуем обработку события нажатия на клавишу в окне Edit1, чтобы запретить ввод любых символов, кроме цифр от 0 до 9. Для создания процедуры обработчика события нажатия на клавишу в окне Edit1 выберите в окне Инспектора объектов компонент Edit1 и на странице Events (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии OnKeyPress. После этого окно Редактора кода немедленно получит фокус и в разделе interface появится запись процедуры обработчика события:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
```

а в разделе implementation — текст заготовки этой процедуры:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
begin
```

```
end;
```

Вставьте в тело процедуры следующий оператор:

```
if not (Key in ['0'.. '9']) then Key:=#0;
```

Действие этого оператора сводится к сравнению значения переменной Key с множеством значений ['0'..'9']. Если символ нажатой клавиши не входит в это множество, то Key присваивается значение нулевого символа (#0). Таким образом, в окне Edit1 будет отображаться текст, состоящий только из цифр.

Создание массива целых чисел опишите в процедуре обработчика события щелчка мышью на кнопке Button1. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект Button1, затем на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика события procedure TForm1.Button1Click(Sender: TObject); в окне Редактора кода следующим образом:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
  Randomize;
  N:=StrToInt(Edit1.Text); {число элементов массива}
  SetLength(M, N);        {задать массиву M длину N}
  Edit2.Text:='';          {очистить окно Edit2}
end;

```

```

for I := 0 to N-1 do      {заполнить массив случайными значениями
                           целых чисел}
begin
  M[I]:= Round(Sin(Random(100))*100);
                           {присвоить элементу массива случайное
                           значение}
  Edit2.Text:=Edit2.Text+' '+IntToStr(M[I]);
                           {вывести элементы массива}
end;
end;

```

ПРИМЕЧАНИЕ

Оператор Randomize; задает случайное начальное значение для функции Random.

Обработку массива опишите в процедуре обработчика события щелчка мышью на кнопке Button2. Для этого выберите в окне Инспектора объектов объект Button2, затем на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода в заготовку процедуры обработчика события procedure TForm1.Button2Click(Sender: TObject); в раздел описания локальных переменных поместите следующее описание:

```

var
  Max, Min, Sum, CountPlus : integer; {результаты обработки массива}

```

где

Max — максимальный элемент массива,

Min — минимальный элемент массива,

Sum — сумма всех элементов массива,

CountPlus — количество положительных элементов массива.

Обработку массива можно реализовать с помощью цикла For, в котором вычисление значения каждой из этих переменных записывается при помощи оператора if then, проверяющим условие CheckBox.Checked. Если свойство Checked имеет значение True, то выполняется вычисление соответствующей переменной. В заключительной части процедуры можно разместить вывод результатов обработки массива. Текст процедуры обработки массива может быть записан следующим образом:

```

procedure TForm1.Button2Click(Sender: TObject); {обработка массива}
var
  Max, Min, Sum, CountPlus : integer; {результаты обработки массива}
begin
  if CheckBox1.Checked then Min:=M[0]; {пусть 0-й элемент - Min}
  Edit3.Text:='';
  if CheckBox2.Checked then Max:=M[0]; {пусть 0-й элемент - Max}
  Edit4.Text:='';
  Sum:=0; {обнулить значения Sum}
  Edit5.Text:='';
  CountPlus:=0; {обнулить значения суммы положительных элементов}
  Edit6.Text:='';

```

```

for I:=0 to N-1 do
begin
  if CheckBox1.Checked then {определить минимальный элемент массива}
    if Min>M[I] then Min:=M[I];
  if CheckBox2.Checked then {определить максимальный элемент массива}
    if Max<M[I] then Max:=M[I];
  if CheckBox3.Checked then {суммировать элементы массива}
    Sum:=Sum+M[I];
  if CheckBox4.Checked then {суммировать положительные
                             элементы массива}
    if M[I]>0 then CountPlus:=CountPlus+1;
end;
{вывести результаты обработки массива}
if CheckBox1.Checked then Edit3.Text:=IntToStr(Min);
if CheckBox2.Checked then Edit4.Text:=IntToStr(Max);
if CheckBox3.Checked then Edit5.Text:=IntToStr(Sum);
if CheckBox4.Checked then Edit6.Text:=IntToStr(CountPlus);
end;

```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Задавая различные значения числа элементов массива и щелкая мышью на кнопке Создать массив, убедитесь в правильной работе процедуры защиты от ввода нечисловых данных в Edit1 и генерации массива случайных целых чисел.

Выбирая варианты обработки массива установкой соответствующих флажков CheckBox, как показано на рис. 13.4, и щелкая мышью на кнопке Вычислить, убедитесь в правильности работы процедуры обработки массива.

Закройте окно приложения.

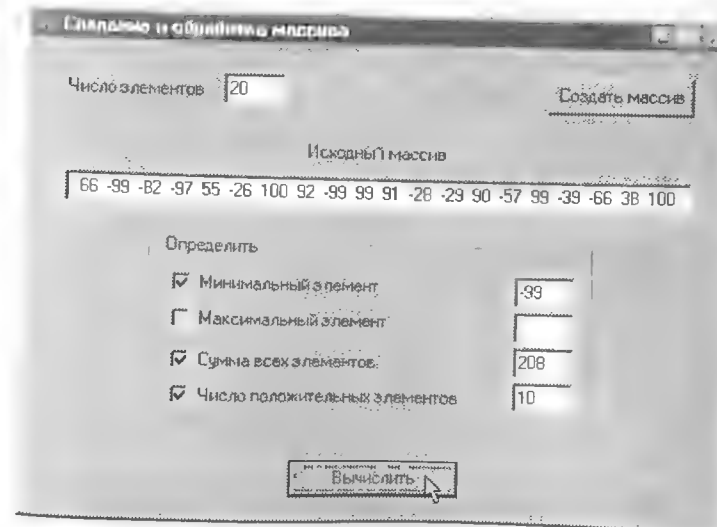


Рис. 13.4. Окно приложения создания и обработки линейного массива

Линейная сортировка массива

Упражнение 4. Создайте приложение, которое предлагает пользователю ввести размер массива и создать массив случайных целых чисел, а затем, используя переключатели, указать порядок сортировки (по неубыванию, по невозрастанию), выполнить сортировку и просмотреть отсортированный массив.

Создайте форму, для свойства **Caption** задайте значение «Линейная сортировка массива». На форме разместите компоненты **Edit1**, **Edit2** и **Edit3**, кнопку **Button1**, для свойства **Caption** которой задайте значение «Создать массив». Удалите текст **Edit1**, **Edit2**, **Edit3** из соответствующих компонентов. Разместите на форме компоненты **Label1**, **Label2** и задайте для их свойств **Caption** значения «Число элементов» и «Исходный массив» соответственно.

Ниже объекта **Edit3** на форме разместите панель **RadioGroup1**, для свойства **Caption** которой задайте значение «Порядок сортировки». Для выбора порядка сортировки задайте два переключателя в панели **RadioGroup1** и подпишите к ним. Выбрав в Инспекторе объектов компонент **RadioGroup1**, на странице свойств выберите свойство **Items**, затем в окне **String List Editor** введите список элементов: По невозрастанию, По неубыванию и нажмите **OK**. Справа от панели **RadioGroup1** разместите кнопку **Button2**, для свойства **Caption** которой задайте значение «Отсортировать».

В нижней части формы разместите **Edit3** для вывода отсортированного массива. Над объектом **Edit3** разместите объект **Label3**, для свойства **Caption** которого задайте значение «Отсортированный массив». Выровняйте компоненты на форме, как показано на рис. 13.5. Зафиксируйте положение компонентов на форме, выбрав в меню **Delphi** команду **Edit ► Lock Controls**.

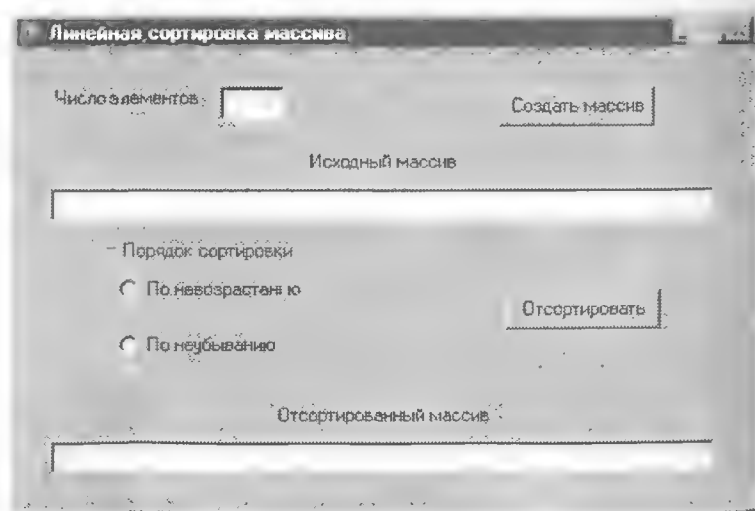


Рис. 13.5. Окно формы приложения линейной сортировки массива

Сохраните файл проекта и программного модуля. Аналогично предыдущему примеру, прежде чем создавать обработчики событий щелчка мышью по кнопкам **Button1** и **Button2**, опишите глобальные переменные целого типа **N** и **I**, где **N** – размер массива, а **I** – порядковый номер элемента массива, а также **M** – динамический массив целых чисел.

```
var
  Form1: TForm1;
  N, I: integer;
  M: array of integer; {описание динамического массива целых чисел}
```

Для предупреждения ошибки ввода в окно **Edit1** нечислового значения введите обработку события нажатия клавиши в окне **Edit1**, чтобы запретить ввод любых символов, кроме цифр от 0 до 9. Для создания процедуры обработчика события нажатия клавиши в окне **Edit1** выберите в окне Инспектора объектов компонент **Edit1** и на странице **Events** (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии **OnKeyPress**. После этого в текст процедуры обработчика события добавьте следующий оператор: **if not (Key in ['0'..'9']) then Key:=#0;**. Полный текст процедуры обработчика события будет выглядеть следующим образом:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9']) then Key:=#0;
end;
```

Создание массива целых чисел опишите в процедуре обработчика события щелчка мышью на кнопке **Button1**. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект **Button1**, затем на странице **События** произведите двойной щелчок на пустом поле списка в событии **OnClick**. После этого в окне Редактора кода в заготовку процедуры обработчика события введите следующий текст:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Randomize;
  N:=StrToInt(Edit1.Text); {число элементов массива}
  SetLength(M, N);        {задать динамическому массиву M длину N}
  Edit2.Text:= '';
  for I:=0 to N-1 do      {заполнить массив случайными значениями целых чисел}
  begin
    M[I]:=Round(Sin(Random(100))*100);
    {присвоить элементу массива случайное значение}
  end;
  Edit2.Text:=Edit2.Text+' '+IntToStr(M[I]);
  {вывести элементы массива}
end;
end;
```

Обработка события нажатия кнопки **Button2** «Вычислить» начинается с сортировки массива, которую можно записать с помощью оператора цикла **for**.

Для хранения номера элемента массива, сравниваемого с текущим (имеющим номер I) введем переменную целого типа J. Для запоминания элемента массива при обмене в процессе сортировки введем переменную Tmp.

Линейная сортировка массива выполняется путем перестановки элементов в массиве при соблюдении следующих условий: если в неотсортированной части массива найден элемент с номером J, больший, чем элемент с номером I (для сортировки по невозрастанию), или меньший, чем элемент с номером I (для сортировки по неубыванию). Поэтому можно записать следующий оператор **if then else** с составным условием:

```
if (RadioGroup1.ItemIndex=0) and (M[I]<M[J])
  or (RadioGroup1.ItemIndex=1) and (M[I]>M[J]) then
```

Перестановка элементов массива осуществляется с использованием третьей переменной:

```
Tmp := M[I];      {запомнить на время значение M[I]}
M[I] := M[J];
M[J] := Tmp;
```

Вывод отсортированного массива в окне Edit3 можно записать оператором:

```
for I:=0 to N-1 do
  Edit3.Text:=Edit3.Text+' '+IntToStr(M[I]);
```

В целом текст процедуры сортировки и вывода отсортированного массива будет выглядеть следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  J, Tmp: integer;
begin
  Edit3.Text:='';
  for I := 0 to N - 2 do {изменять размер неотсортированной
    части массива}
    for J:=I+1 to N-1 do {сравниваем поочередно I-й элемент
      неотсортированной части массива со всеми
      от I+1-го до конца}
      begin
        {выбор операции в зависимости от значения
        свойства RadioGroup1.ItemIndex}
        if (RadioGroup1.ItemIndex=0) and (M[I]<M[J])
          or (RadioGroup1.ItemIndex=1) and (M[I]>M[J]) then
          {если в неотсортированной части массива нашли
          J-й элемент, больший чем I-й (для сортировки
          по невозрастанию) или меньший чем I-й
          (для сортировки по неубыванию)}
          begin
            {обменять местами элементы массива}
            Tmp := M[I]; {запомнить на время значение M[I]}
            M[I] := M[J];
            M[J] := Tmp;
          end;
        end;
    end;
```

```
for I:=0 to N-1 do {вывести отсортированный массив}
  Edit3.Text:=Edit3.Text+' '+IntToStr(M[I]);
```

end;

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Задавая различные значения числа элементов массива и щелкая мышью на кнопке Создать массив, создавайте линейный массив целых чисел. Выбирая при помощи переключателей в панели RadioGroup1 вариант сортировки и щелкая мышью на кнопке Отсортировать, как показано на рис. 13.6, убедитесь в правильной работе процедуры сортировки и вывода отсортированного массива.

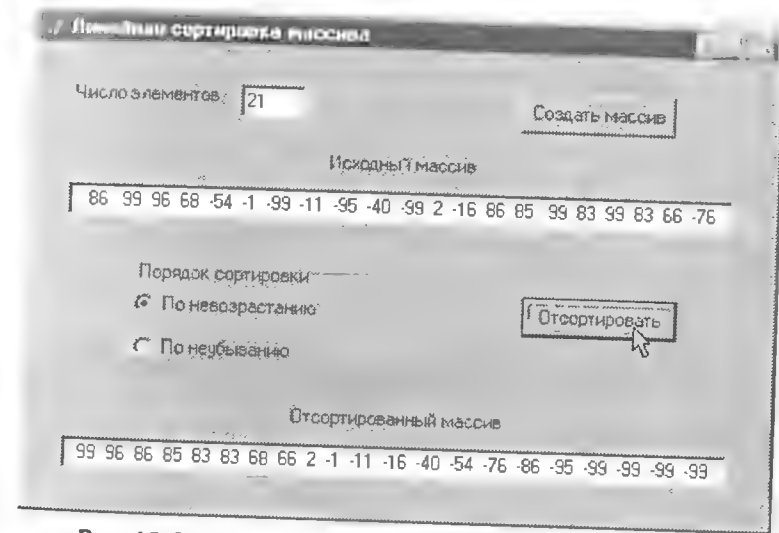


Рис. 13.6. Окно приложения линейной сортировки массива

После окончания проверки работы приложения закройте его окно.

Использование компонента StringGrid для представления двумерных массивов

Упражнение 5. Создайте приложение, которое выводит двумерный массив случайных целых чисел в объекте StringGrid и определяет минимальный и максимальный элементы, а также сумму элементов массива, расположенных на главной диагонали.

Создайте форму, свойству Caption которой присвойте значение «Обработка двумерного массива». Выберите в палитре компонентов страницы Additional компонент StringGrid и разместите его в левом верхнем углу формы. Задайте для свойств ColCount (количество столбцов) и RowCount (количество строк)

значения 6. Задайте для свойств FixedCols и FixedRows (количество фиксированных, непрокручиваемых столбцов и строк, используемых для размещения надписей номеров столбцов и строк).

ПРИМЕЧАНИЕ

Компонент StringGrid представляет собой таблицу, содержащую строки. Таблица может иметь полосы прокрутки, причем заданное число первых строк и столбцов может быть фиксированным и не подвергаться прокрутке. Таким образом, можно задать заголовки столбцов и строк, постоянно присутствующие в окне компонента. Каждой ячейке таблицы может быть поставлен в соответствие некоторый объект.

Справа от объекта StringGrid разместите кнопку Button1 и задайте для ее свойства Caption значение «Заполнить». Ниже объекта StringGrid разместите панель GroupBox1 и присвойте ее свойству Caption значение «Определить». На данной панели разместите компоненты CheckBox1, CheckBox2, CheckBox3 и назначьте свойству Caption значения «Минимальный элемент», «Максимальный элемент», «Сумма элементов главной диагонали», соответственно. Справа от компонентов CheckBox1, CheckBox2, CheckBox3 разместите компоненты Edit1, Edit2, Edit3 и удалите текст Edit1, Edit2, Edit3 из соответствующих компонентов. Правее панели GroupBox1 разместите кнопку Button2 и задайте для ее свойства Caption значение «Вычислить». Выводите компоненты на форме, как показано на рис. 13.7, и зафиксируйте их положение на форме.

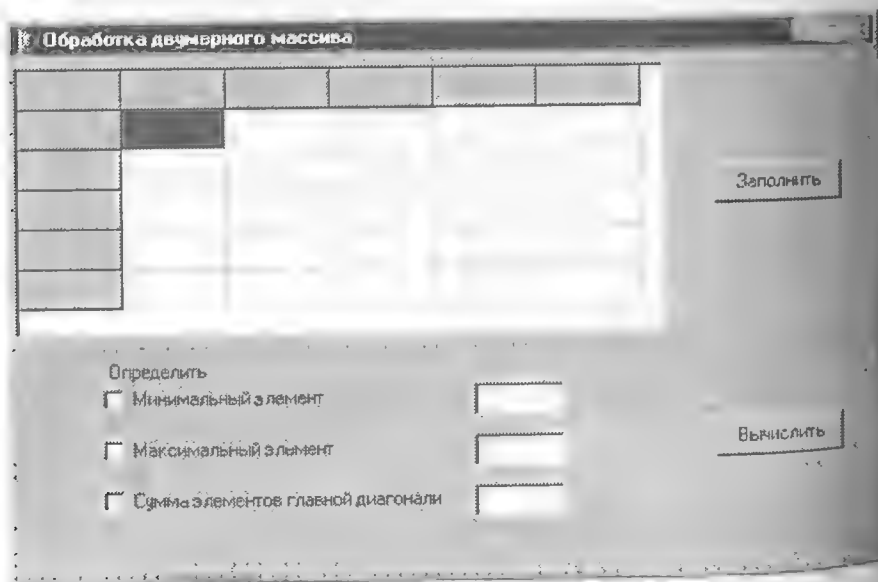


Рис. 13.7. Окно формы приложения обработки двумерного массива

Сохраните файл проекта и программного модуля. Аналогично предыдущему примеру, прежде чем создавать обработчики событий щелчка мышью по

кнопкам Button1 и Button2, следует добавить в раздел описания переменных данного модуля целочисленные переменные I и J, предназначенные для хранения индексов массива (I — номер столбца, J — номер строки).

```
var
  Form1: TForm1;
  I, J: Integer;
```

Для получения подсказки Delphi по объекту StringGrid1, указав объект, нажмите F1. В окне Delphi Help просмотрите общую информацию о назначении объекта. Щелкнув мышью по ссылке Properties (Свойства), откройте окно со списком свойств объекта, и, выбирая нужные свойства, например, Cells, ColCount и т. п., просмотрите справочную информацию. Для возврата к предыдущему экрану справки воспользуйтесь кнопкой Назад в панели инструментов окна Delphi Help. Щелкая мышью по ссылкам Methods (Методы) и Events (События), просмотрите список методов и событий объекта. Для просмотра примеров следует щелкнуть мышью по ссылке Example. Завершив просмотр справочной информации, закройте окно Delphi Help.

ВНИМАНИЕ

Обратите внимание на использование оператора with .. do для обращения к элементам объекта. При изучении языка Turbo Pascal вы использовали этот оператор для сокращения записи при обращении к полям записей. Теперь вы будете применять его для сокращения записи при обращении к свойствам и методам объекта. Это делается, чтобы избежать повторных ссылок на объект в последующих операторах. В операторе, следующем за ключевым словом do, можно для полей, свойств и методов объекта не указывать ссылки на этот объект. При этом каждый идентификатор в операторе, совпадающий с именем поля, свойства, метода объекта, трактуется как относящийся к этому объекту, и к нему неявно добавляется ссылка на этот объект. Например, вместо того чтобы, обращаясь к ячейке объекта StringGrid1, писать StringGrid1.Cells[I,J], удобнее использовать оператор with StringGrid1 do, в теле которого можно неоднократно обращаться к Cells[I,J], не упоминая имени объекта StringGrid1.

Создайте процедуру обработчика события щелчка мышью на кнопке Button1, в которой сначала будет выполнена операция вывода номеров строк и столбцов, а затем ячейки StringGrid1 будут заполнены случайными целыми числами. Для этого выберите в окне Инспектора объектов объект Button1 и на странице События произведите двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Randomize;
  with StringGrid1 do
  begin
    I:=0;
    {заполнение массива}
    {вывести номера строк в 0-м столбце
    и столбцов в 0-й строке}
    {столбец 0}
```

```

for J:= 1 to RowCount - 1 do {вывести номера строк}
  Cells[1,J] := IntToStr(J);
J:=0; {строка 0}
for I:= 1 to ColCount - 1 do {вывести номера столбцов}
  Cells[I,J] := IntToStr(I);
end;
with StringGrid1 do {вывести в таблице элементы двумерного массива}

for I := 1 to ColCount - 1 do
  for J:= 1 to RowCount - 1 do
    begin
      Cells[I,J] := IntToStr(Round(Sin(Random(100))*100));
    end;
  end;
end;

```

Для создания процедуры обработки массива выберите в окне Инспектора объектов объект Button2 и на странице События произведите двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TForm1.Button2Click(Sender: TObject); {обработка массива}
var
  Min, Max, Sum : Integer; {локальные переменные - результаты
                           {обработки массива}
begin
  if Checkbox1.Checked then {определение Min-элемента}
    with StringGrid1 do
      begin
        Min:=StrToInt(Cells[1,1]); {пусть - это Min-элемент}
        for I := 1 to ColCount - 1 do
          for J:= 1 to RowCount - 1 do
            if StrToInt(Cells[I,J])<Min then
              Min:=StrToInt(Cells[I,J]);
        Edit1.Text:=IntToStr(Min);
      end
    else Edit1.Text:='';
    if Checkbox2.Checked then {определение Max-элемента}
      with StringGrid1 do
        begin
          Max:=StrToInt(Cells[1,1]); {пусть - это Max-элемент}
          for I := 1 to ColCount - 1 do
            for J:= 1 to RowCount - 1 do
              if StrToInt(Cells[I,J])>Max then
                Max:=StrToInt(Cells[I,J]);
          Edit2.Text:=IntToStr(Max);
        end
      else Edit2.Text:='';
      if Checkbox3.Checked then {вычисление Sum}
        with StringGrid1 do

```

```

begin
  Sum:=0; {обнулить значение суммы перед подсчетом}
  for I := 1 to ColCount - 1 do
    Sum:=Sum+StrToInt(Cells[I,1]);
  Edit3.Text:=IntToStr(Sum);
end else Edit3.Text:='';

```

end;

Как видно из текста процедуры, в ней имеется три фрагмента, каждый из которых выполняет вычисления, если свойство Checked соответствующего флажка (Checkbox1, Checkbox2 или Checkbox3) имеет значение True. Операторы типа Edit1.Text:=""; Edit2.Text:=""; Edit3.Text:=""; обеспечивают очистку соответствующего окна Edit, если вычисление не проводилось. Для обращения к элементу массива, расположенному на главной диагонали StringGrid, указывается одинаковый номер строки и столбца Cells[I,I].

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Щелкнув мышью на кнопке Заполнить, проверьте заполнение надписей номеров строк и столбцов, а также заполнение объекта StringGrid значениями элементов двумерного массива. Как показано на рис. 13.8, устанавливая флажки Checkbox1..3 и щелкая мышью на кнопке Вычислить, убедитесь, что приложение правильно обрабатывает двумерный массив: определяет минимальное и максимальное значение, вычисляет сумму элементов на главной диагонали.

После проверки работы приложения закройте его окно.

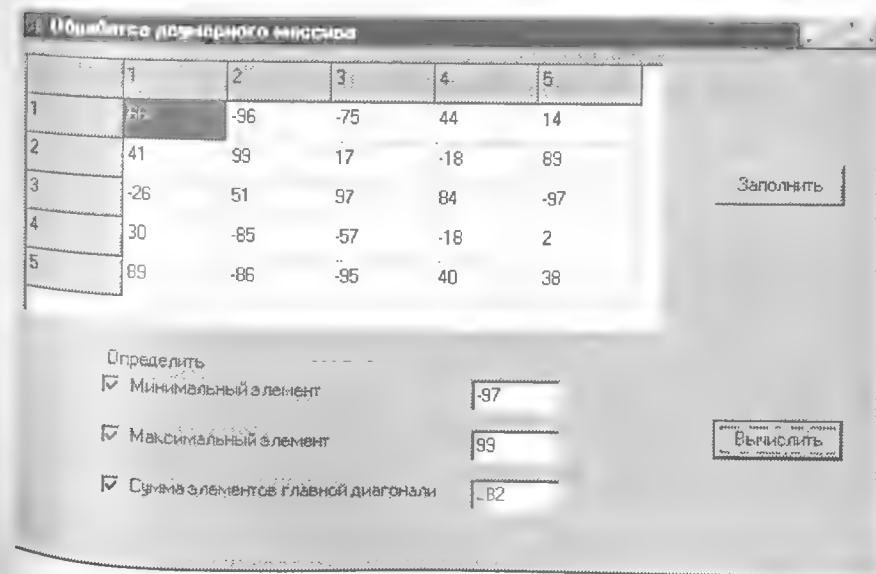


Рис. 13.8. Окно приложения обработки двумерного массива

Упражнение 6. Создайте приложение, осуществляющее вывод в окно двумерного массива — табеля успеваемости по нескольким школьным предметам с оценками за 1–4 четверти и за год, причем годовая оценка должна вычисляться как среднее арифметическое всех оценок по данному предмету за 1–4 четверти. Используйте закраску ячеек таблицы разными цветами, которые могут определяться как на стадии разработки формы приложения, так и во время работы приложения, в зависимости от значения элемента массива.

Создайте форму, свойству **Caption** которой присвойте значение «Табель успеваемости». Выберите в палитре компонентов страницы **Additional** компонент **StringGrid** и разместите его в левом верхнем углу формы. В окне Инспектора объектов выберите компонент **StringGrid1** и задайте для его свойства **Align** (Выравнивание) значение **alClient** (Вся клиентская область формы). При этом компонент **StringGrid1** должен занимать всю клиентскую область формы и во время выполнения приложения его размеры должны изменяться при изменении размеров формы. Задайте для свойств **ColCount** (количество столбцов) и **RowCount** (количество строк) значения 6. Задайте для свойства **FixedRows** (количество фиксированных, не прокручиваемых строк, используемых для размещения надписей номеров столбцов и строк) значение 1. Сохраните файл проекта и программного модуля.

Создание табеля с оценками опишите в процедуре обработчика события создания формы **Form1**. Для этого выберите в окне Инспектора объектов объект **Form1** и на странице События произведите двойной щелчок мышью на пустом поле списка в событии **OnCreate**, наступающем при создании формы. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события **procedure TForm1.FormCreate(Sender: TObject);**

Вставьте в текст модуля **main** перед разделом **implementation** описание раздела констант и переменных, задающих постоянные параметры массива и столбцов **StringGrid1**:

```
const
  COLWIDTHS = array[0..5] of integer=(85,70,70,70,70,60);
                                     {ширина столбцов таблицы}
  COLTITLES = array[0..5] of string=('Предмет', '1-я четверть', '2-я четверть', '3-я
четверть', '4-я четверть', 'годовая');
                                     {шапка таблицы}
  COLALIGNS = array[0..5] of byte=(1,2,2,2,2,2); {вариант выравнивания}
var
  Form1: TForm1;                      {объект Form1}
  ColAll: integer;                    {ширина курсора}
```

Отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TForm1.FormCreate(Sender: TObject);
var I,J: integer;
    K : real;
```

```
begin
  for I:=0 to 5 do
  begin
    Inc(ColAll, COLWIDTHS[I]+1);
    {Подсчет общей ширины для курсора}
    Grid1.ColWidths[I] := COLWIDTHS[I];
    {Установка ширины колонок}
    StringGrid1.Cells[ 1,0 ] := COLTITLES[I];
    {Установка шапки таблицы}
  end;
  with StringGrid1 do begin
    {Массив с данными успеваемости - 0-й столбец}
    Cells[ 0, 1]:='Русский язык';
    Cells[ 0, 2]:='Математика';
    Cells[ 0, 3]:='История';
    Cells[ 0, 4]:='География';
    Cells[ 0, 5]:='Музыка';

    {Оценки за 1-ю четверть - 1-й столбец}
    Cells[ 1, 1]:='5';
    Cells[ 1, 2]:='4';
    Cells[ 1, 3]:='4';
    Cells[ 1, 4]:='4';
    Cells[ 1, 5]:='5';

    {Оценки за 2-ю четверть - 2-й столбец}
    Cells[ 2, 1]:='4';
    Cells[ 2, 2]:='2';
    Cells[ 2, 3]:='4';
    Cells[ 2, 4]:='5';
    Cells[ 2, 5]:='3';

    {Оценки за 3-ю четверть - 3-й столбец}
    Cells[ 3, 1]:='5';
    Cells[ 3, 2]:='2';
    Cells[ 3, 3]:='3';
    Cells[ 3, 4]:='4';
    Cells[ 3, 5]:='5';

    {Оценки за 4-ю четверть - 4-й столбец}
    Cells[ 4, 1]:='4';
    Cells[ 4, 2]:='3';
    Cells[ 4, 3]:='4';
    Cells[ 4, 4]:='5';
    Cells[ 4, 5]:='5';

    {5-й столбец: вычисление оценок за год
    как среднего арифметического оценок
    за 4 учебные четверти}
```

```
for J:=1 to 5 do
begin
  K:=0;
  for I:=1 to 4 do
```



```

K:=K+StrToInt(Cells[J,I]):
K:=K/4; {вычисление среднего балла за год}
Cells[J,I]:=FloatToStr(K):
end:
end:
end:

```

Как видно из текста процедуры, сначала задаются параметры «шапки» таблицы, затем задаются элементы массива: названия предметов и оценки за 1–4 четверти. В заключительном фрагменте процедуры выполняется вычисление оценки за год как среднего арифметического оценок по предмету за все четверти.

Для раскраски ячеек StringGrid1 создайте процедуру обработчика события OnDrawCell, для чего выберите в окне Инспектора объектов объект StringGrid1 на странице Свойства произведите двойной щелчок мышью на пустом поле списка в событии OnDrawCell. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col: Integer; Row: Integer; Rect: TRect; State: TGridDrawState);
var
  CurrCell:String; {текущая ячейка}
begin
  with TStringGrid(sender).Canvas.Rect do
  begin
    CurrCell:=TStringGrid(sender).Cells[Col,Row]:
    Bottom:=Bottom+1;
    Font.Color:=clBlack; {цвет шрифта}
    Brush.Color:=clWhite; {цвет заливки}
    if Row<1 then begin {если нулевая строка, то рисуем шапку таблицы}
      Brush.Color:=clBtnFace; {цвет заливки - текущий цвет кнопок}
      Bottom:=Bottom-1;
      DrawEdge(Handle, Rect, BDR_RAISEDINNER, BF_TOPRIGHT or BF_MIDDLE or BF_BOTTOMLEFT);
      InflateRect(Rect, -2, -2);
      TextRect(Rect, Left-2+((COLWIDTHS[Col]-TextWidth(CurrCell))shr 1), Top-2, CurrCell);
    end;
    exit;
  end;
  case Col of
    0: brush.Color:=clBlue; {закраска ячеек в разных столбцах}
    1..4: brush.Color:=clAqua; {первый - четвертый столбцы - голубые}
    5: {цвет заливки пятого столбца и цвет шрифта символов зависит от значения ячейки}
  end;
  if StrToFloat(CurrCell)<3 then
  begin
    brush.Color:=clRed; {если оценка <3, то цвет заливки красный}
  end;
end;

```

```

Font.Color:=clAqua; {цвет шрифта символов голубой}
end
else
  if StrToFloat(CurrCell)<4 then
  begin
    brush.Color:=clYellow; {если оценка <4, то цвет заливки желтый}
    Font.Color:=clOlive; {цвет шрифта символов оливковый}
  end
else
  begin
    brush.Color:=clGreen; {иначе - цвет заливки зеленый}
    Font.Color:=clYellow; {цвет шрифта символов желтый}
  end;
end;
end;
{выравнивание значений в столбцах(COLALIGN)}
case COLALIGN[Col] of
  1: {влево} TextRect(Rect, Left, Top, CurrCell);
  2: {по центру} TextRect(rect, Left+((COLWIDTHS[Col]-TextWidth(CurrCell)) shr 1), Top, CurrCell);
end;
{рисование курсора выделения строки}
if (Row=TStringGrid(sender).Row) then
begin
  Left:=0; {левая граница}
  Bottom:=Bottom-1; {верхняя граница}
  Right:=ColAll; {правая граница}
  Brush.Color:=clBlack; {цвет}
  TStringGrid(sender).Canvas.FrameRect(Rect); {курсор в таблице}
end;
end;
end;

```

Изучите текст процедуры и комментарии к ее фрагментам. Сохраните файлы модуля и проекта, откомпилируйте и запустите приложение. Окно приложения будет выглядеть следующим образом:

Предмет	1-я четверть	2-я четверть	3-я четверть	4-я четверть	годовая
Математика	4	2	2	3	2.75
Русский язык	4	4	3	4	3.75
История	4	5	4	5	4.5
Физика	5	3	5	5	4.5

Рис. 13.9. Окно приложения «Табель успеваемости»

Проверьте работу приложения. Закройте окно приложения и задайте другие значения элементов массива, например, изменив оценки за четвертую четверть, как показано ниже:

```
Cells[ 4, 1]:='5';
Cells[ 4, 2]:='4';
Cells[ 4, 3]:='3';
Cells[ 4, 4]:='4';
Cells[ 4, 5]:='3';
```

Сохраните изменения в проекте, перекомпилируйте и запустите приложение. Обратите внимание, что с изменением значения среднего балла за год по предметам поменялся цвет закрашки и шрифта символов текста.

Закройте окно приложения.

Ввод и обработка элементов массива с использованием StringGrid

Упражнение 7. Создайте приложение с компонентом StringGrid, обеспечивающее пользователю возможность ввода значений элементов двумерного массива в ячейки и выполняющее вычисление суммы элементов массива.

Создайте форму, для свойства Caption задайте значение «Ввод и суммирование элементов массива». Выберите в палитре компонентов страницы Additional компонент StringGrid и разместите его по центру формы. В окне Инспектора объектов выберите компонент StringGrid1 и присвойте его свойству Align (Выравнивание) значение alNone (Не изменять). При этом компонент StringGrid1 остается там, куда он был помещен во время проектирования, и размеры его не изменяются. Задайте для свойств ColCount и RowCount значения 5. Задайте для свойств FixedCols и FixedRows (количество фиксированных, не прокручиваемых столбцов и строк, используемых для размещения надписей номеров столбцов и строк) значение 1. На странице свойств Инспектора объектов разверните список свойства StringGrid1.Options и задайте для его под-свойств: goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect, goDrawFocusSelected и goEditing значение True.

В левом нижнем углу формы разместите кнопку Button1 и присвойте ее свойству Caption значение «Вычислить». В правом нижнем углу формы разместите объект Edit1 для отображения суммы элементов массива. Над объектом StringGrid1 разместите компонент Label1 и задайте для его свойства Caption значение «Введите элементы массива». Слева от объекта Edit1 разместите компонент Label2 и задайте для его свойства Caption значение «Сумма элементов». Удалите текст Edit1 из окна Edit1. Выводите компоненты, как показано на рис. 13.10, и зафиксируйте их положение на форме.

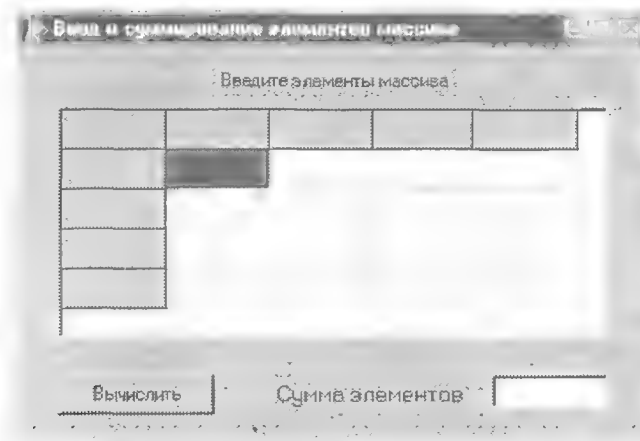


Рис. 13.10. Окно формы приложения ввода и суммирование элементов массива

Сохраните файл проекта и программного модуля.

Чтобы обеспечить пользователю возможность ввода значений элементов двумерного массива в ячейки StringGrid1, создайте процедуру обработчика события OnEnter для объекта StringGrid1. Для этого выберите в окне Инспектора объектов объект StringGrid1 и на странице События произведите двойной щелчок на пустом поле списка в событии OnEnter, наступающем в момент получения элементом фокуса. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.FormCreate(Sender: TObject);

Так как эта процедура сводится к выводу в окне приложения таблицы StringGrid1 подписи номеров строк и столбцов, то текст этой процедуры можно записать следующим образом:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I, J: Integer; {номера столбцов и строк}
begin
  with StringGrid1 do {вывести номера строк в 0 столбце
    begin {и столбцов в 0 строке}
      I := 0; {столбец 0}
      for J := 1 to RowCount - 1 do
        Cells[I, J] := IntToStr(J);
      J := 0; {строка 0}
      for I := 1 to ColCount - 1 do
        Cells[I, J] := IntToStr(I);
      end;
    end;
```

Для предупреждения ошибки ввода в окно ячейки StringGrid1 нечислового значения введите обработку нажатия клавиши, чтобы запретить ввод любых

символов, кроме цифр от 0 до 9 и знаков «-» и «+». Для создания процедуры обработчика события нажатия клавиши выберите в окне Инспектора объектов компонент `StringGrid1` и на странице Events (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии `OnKeyPress`. После этого окно Редактора кода немедленно получит фокус и в разделе `interface` появится запись процедуры обработчика события: `procedure TForm1.StringGrid1.KeyPress(Sender: TObject; var Key: Char);`

Вставьте в текст заготовки этой процедуры оператор

```
if not (Key in ['0'..'9', '-', '+']) then Key:=#0;
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Как вы увидите, приложение обеспечивает ввод числовых значений элементов массива в ячейки `StringGrid1`, но не вычисляет их сумму при щелчке мышью на кнопке `Button1`, так как процедура обработки этого действия еще не создана.

Для подсчета элементов введенного массива следует записать в виде процедуры обработчик события щелчка мышью на кнопке `Button1`. Для генерации кода процедуры обработчика события выберите в окне Инспектора объектов объект `Button1` и на странице События произведите двойной щелчок на пустом поле списка в событии `OnClick`, наступающем в момент получения элементом фокуса. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`, в которую следует добавить следующие операторы:

```
procedure TForm1.Button1Click(Sender: TObject);
    {подсчет суммы элементов}
var
    I, J, Sum: integer;
begin
    with StringGrid1 do
    begin
        Sum:=0; {обнулить сумму перед суммированием
                элементов}

        Edit1.Text:='';
        for J:=1 to RowCount - 1 do
            for I:=1 to ColCount - 1 do
                if Cells[J,I]<>'' then
                    {если ячейка заполнена,
                     то суммировать ее значение}
                    Sum:=Sum+StrToInt(Cells[J,I]);
        Edit1.Text:=IntToStr(Sum); {вывод результата суммирования}
    end;
end;
```

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Теперь оно обеспечивает не только ввод числовых значений элементов массива в ячейки `StringGrid1`, но и вычисляет их сумму при щелчке на кнопке `Button1`, как показано на рис. 13.11.

Закройте окно приложения.

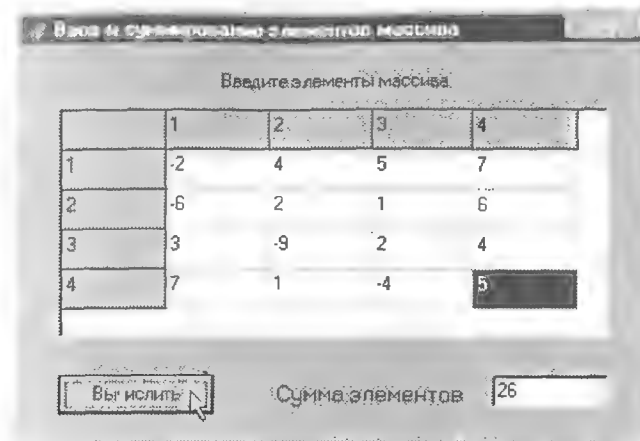


Рис. 13.11. Окно приложения ввода и суммирования элементов массива

Обработка файлов

Упражнение 8. Создайте приложение, которое создает текстовый файл `text1.txt` и записывает в него текст, введенный пользователем в окно `Edit`, после чего закрывает файл.

Создайте форму и задайте для ее свойства `Caption` значение «Создание файла и вывод в него текста из Edit». Разместите на форме компоненты `Edit1`, `Label1` и `Button1`, как показано на рис. 13.12. Задайте значения для свойств `Label1.Caption` — «Введите текст», `Button1.Caption` — «Сохранить». Удалите текст `Edit1` из соответствующего компонента. Выровняйте компоненты и зафиксируйте их положение на форме.

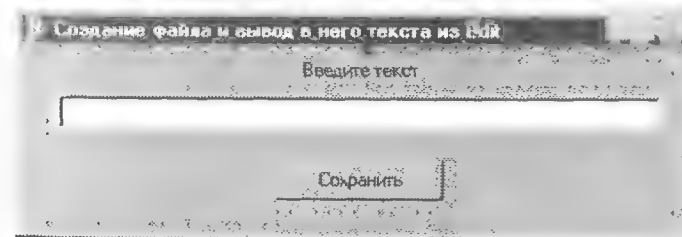


Рис. 13.12. Окно формы

Сохраните файлы модуля (под именем `main`) и проекта (под именем `TextEditFile`) в папке Обработка текстовых файлов. Откомпилируйте приложение и запустите его на выполнение. Убедитесь в том, что приложение позволяет пользователю вводить текст в окно `Edit1`. Для того чтобы на диске создавался

текстовый файл, и текст из окна Edit1 записывался в него, создайте процедуру обработчика события щелчка мышью на кнопке Button1 «Сохранить» и в ней реализуйте создание нового файла и запись в него текста. Для создания процедуры обработчика щелчка мышью на кнопке Button1 выберите в окне Инспектора объектов объект Button1, затем на странице Событий произведите двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Дополните текст процедуры следующим кодом:

```
var
  f:TextFile;           {описание файловой переменной}
begin
  AssignFile(f,'text1.txt'); {связь файловой переменной с файлом}
  Rewrite(f);             {создать новый файл}
  WriteLn(f,Edit1.Text);  {записать в файл}
  CloseFile(f);          {закрыть файл}
end;
```

Как видно из текста процедуры, знакомого вам по аналогичной теме из первой части книги об использовании среды программирования Turbo Pascal, в начале процедуры файловая переменная `f` связывается с именем файла `text1.txt`, а затем на диске создается новый файл с этим именем. После этого выполняется запись значения свойства `Text` объекта `Edit1`, и файл закрывается. Сохраните файл модуля под именем `Main`, файл проекта — под именем `TextEditFile1`, откомпилируйте программу и запустите ее на выполнение.



Рис. 13.13. Окно приложения, сохраняющего текст из Edit1 в файл

В окне приложения введите текст, как показано на рис. 13.13, и щелкните мышью на кнопке Сохранить. Закройте окно приложения и откройте в окне Проводника Windows папку Обработка текстовых файлов, в которой сохранены файлы проекта. В списке файлов этой папки вы увидите и вновь созданный файл `text1.txt`, как показано на рис. 13.14.

Дважды щелкните левой кнопкой мыши на имени файла `text1.txt`. Будет открыто окно редактора Блокнот с этим файлом. Убедитесь, что это — тот самый текст, который вы только что вводили в окне приложения. Закройте окна редактора Блокнот и Проводника Windows.

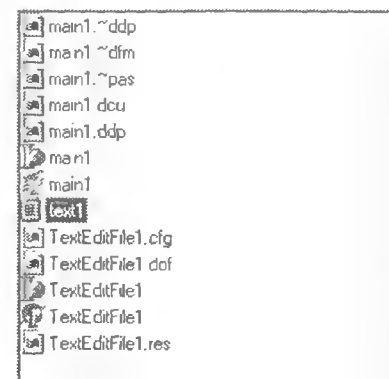


Рис. 13.14. Файл `text1.txt` в папке проекта

Упражнение 9. Создайте приложение, открывающее текстовый файл для чтения и считывающее из него текст в окно Мемо. Перед открытием файла следует проверить его наличие; в случае отсутствия файла должно выводиться соответствующее сообщение.

ПРИМЕЧАНИЕ

Многострочное окно редактирования Мемо используется для ввода, отображения и редактирования многострочных текстов. Свойство `Lines`, доступное как во время проектирования, так и во время выполнения, имеет большое количество свойств и методов типа `Strings`, которые обычно используются для формирования и редактирования текста. Весь текст содержится в свойстве `Text`.

Создайте форму и задайте для ее свойства `Caption` значение «Чтение текста из файла в окно Мемо». На форме `Form1` разместите компоненты `Memo1`, `Label1` и `Button1`, как показано на рис. 13.12, задайте значения для свойств `Label1.Caption` — «Текст из файла», `Button1.Caption` — «Прочитать текст из файла». Для удаления текста `Memo1` из окна компонента `Memo1` выберите в окне Инспектора объектов объект `Memo1`, затем на странице Свойства произведите двойной щелчок на поле значения `Strings` свойства `Lines` для формирования и редактирования текста. После этого в окне `String List Editor` удалите текст `Memo1`, как показано на рис. 13.15, и щелкните мышью на кнопке `OK`.

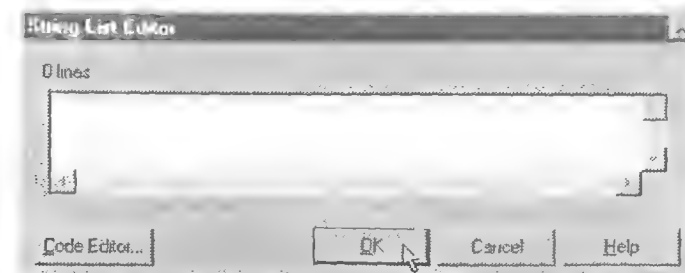


Рис. 13.15. Редактирование текста в окне `Memo1`

ПРИМЕЧАНИЕ

Директивы компилятора \$I позволяют включить или отключить автоматический контроль результата вызова процедур ввода-вывода Object Pascal. Если используется директива {\$I+}, то при возвращении процедурой ввода-вывода ненулевого значения генерируется исключение EInOutError и в его свойство errorCode заносится код ошибки. Таким образом, при использовании директивы {\$I+} операции ввода-вывода располагаются в блоке try...except, имеющем обработчик исключения EInOutError. Если такого блока нет, то обработка производится методом TApplication.HandleException. Если используется директива {\$I-}, то исключение не генерируется. В этом случае можно проверить наличие ошибки, обратившись к функции IOResult. Эта функция возвращает код ошибки, который можно проанализировать.

Сохраните файл модуля под именем main1, а файл проекта — под именем TextMemoFile1 в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув на кнопке Прочитать текст из файла, убедитесь, что текст из файла считывается в окно Memo1, как показано на рис. 13.16.

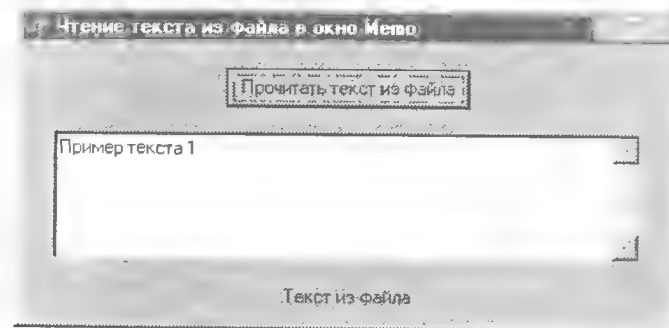


Рис. 13.16. Вид окна приложения чтения текста из файла в окно Мемо

Закройте окно приложения.

Упражнение 10. Создайте приложение, открывающее текстовый файл для дополнения и затем добавляющее в него введенный текст.

Создайте форму и задайте для ее свойства Caption значение «Добавление текста в файл». На форме Form1 разместите компоненты Button1, Memo1, Label1 и, как показано на рис. 13.17, присвойте значения свойствам Label1.Caption — «Текст из файла», Button1.Caption — «Прочитать текст из файла». Под Label1 разместите Label2 и задайте для свойства Label2.Caption значение «Добавляемый текст». Ниже Label2 разместите на форме Edit1. Под объектом Edit1 разместите кнопку Button2 и задайте для свойства Button1.Caption значение «Добавить текст в файл». Удалите текст из окон компонентов Memo1, Edit1. Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектора объектов выберите объект Memo1 и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте компоненты и зафиксируйте их положение на форме, как показано на рис. 13.17.

Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектора объектов выберите объект Memo1 и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте компоненты и зафиксируйте их положение на форме.

Выбрав в окне Инспектора объектов объект Button1, на странице Событий произведите двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события. Чтобы эта процедура выполняла открытие текстового файла и выводила текст в окно Memo1, отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F:TextFile; {описание файловой переменной}
  Ch : Char;   {описание переменной, в которую будет считываться
               символ из файла}
begin
  AssignFile(F,'text1.txt'); {связь файловой переменной с файлом
                             на диске}
  {$I-} {директива компилятора: отключить
        проверку ошибок ввода вывода}
  Reset(F); {открыть файл для чтения}
  {$I+}
  if IOResult = 0 then begin {если операция открыть файл
                             выполнена успешно}
    while not Eof(f) do {пока не конец файла}
    begin
      Read(F,Ch); {прочитать из файла символ}
      Memo1.Text:=Memo1.Text+Ch; {вывести символ в поле Memo1}
    end;
    CloseFile(F); {закрыть файл}
  end
  else {IOResult<>0 - операция открыть файл не выполнена}
    ShowMessage('Нет такого файла');
end;
```

Как видно из текста процедуры, после связывания файловой переменной с именем файла при помощи специальной директивы компилятора отключается контроль ошибок ввода-вывода для того, чтобы проверить функцией IOResult успешность операции. В случае успешности операции открытия файла на чтение производится чтение файла по одному символу и вывод его в окно Memo1. Условием прекращения чтения из файла является достижение конца файла (Eof(f)=True). После этого файл закрывается. Если открыть файл не удастся, то IOResult возвращает значение, отличное от нуля, при этом в отдельном окне выводится соответствующее сообщение.

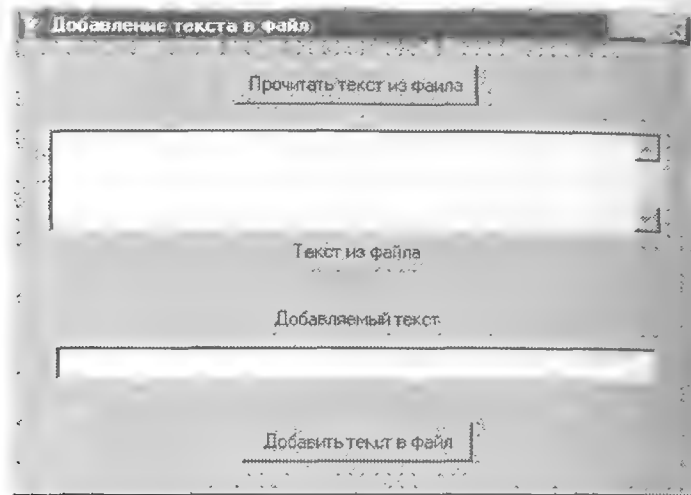


Рис. 13.17. Окно формы

Создайте обработчики нажатий кнопок Прочитать текст из файла и Добавить текст в файл. Процедура обработки события нажатия кнопки Прочитать текст из файла была описана в упражнении 9. Для создания процедуры обработчика события нажатия кнопки Добавить текст в файл выберите в окне Инспектора объектов объект `Button2`, на странице События произведите двойной щелчок на пустом поле списка в событии `OnClick`. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события. Чтобы эта процедура выполняла открытие существующего текстового файла и добавляла в него текст, являющийся значением свойства `Text` объекта `Edit1`, отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  f:TextFile;           {описание файловой переменной}
begin
  AssignFile(f,'text1.txt');
  Append(f);            {открыть существующий файл для добавления
                        в его конец}
  WriteLn(f,Edit1.Text); {записать в файл значение свойства Text
                        объекта Edit1}
  CloseFile(f);         {закрыть файл}
end;
```

Сохраните файл модуля под именем `Main2`, а файл проекта — под именем `TextMemoFile2` в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув мышью на кнопке Прочитать текст из файла, убедитесь, что текст из файла на диске считывается в окно `Memo1`, как показано на рис. 13.18.

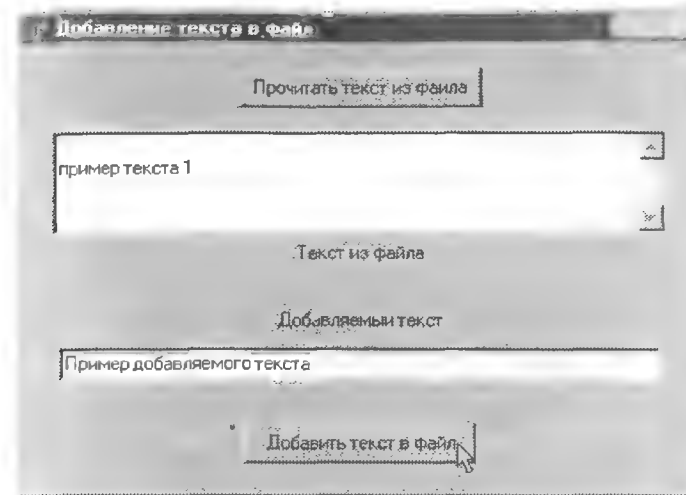


Рис. 13.18. Окно приложения, добавляющего текст в файл

Введите текст в окно `Edit1` и щелкните мышью на кнопке Добавить текст в файл. Щелкнув на кнопке Прочитать текст из файла, просмотрите текст, прочитанный из файла, и убедитесь, что при повторном считывании из файла считывается текст, включающий текст, только что добавленный вами. Закройте окно приложения.

Упражнение 11. Создайте приложение, создающее текстовый файл и сохраняющее в нем текст из `Memo`, используя метод `SaveDialog` для выбора папки и задания имени и расширения текстового файла.

Создайте форму, на которой разместите компоненты `Memo1`, `Label1` и `Button1`. Задайте для свойства `Form1.Caption` значение «Пример с использованием `SaveDialog`», для `Label1.Caption` — значение «Введите текст», для `Button1.Caption` — значение «Сохранить». Удалите текст «`Memo1`» из окна `Memo1`. Выровняйте и зафиксируйте визуальные компоненты на форме, как показано на рис. 13.19. Выберите в палитре компонентов страницу `Dialogs` и поместите на форму компонент `SaveDialog`. Так как он не является визуальным компонентом, то его можно поместить в любое место формы. Задайте для свойства `SaveDialog1.Title` значение «Сохранить текстовый файл», которое будет отображаться в заголовке диалогового окна сохранения файла.

Чтобы при сохранении файла в окне диалога обеспечить выбор типа файла, выберите в окне Инспектора объектов объект `SaveDialog1`, на странице Свойства произведите двойной щелчок в списке значений свойства `Filter`. В окне `Filter Editor`, как показано на рис. 13.20, задайте фильтры для выбора типа и расширения файла.

Щелкнув на кнопке `OK`, закройте окно `Filter Editor`. Для установки расширения `*.txt` по умолчанию задайте значение 1 для свойства `SaveDialog1.Filter`.

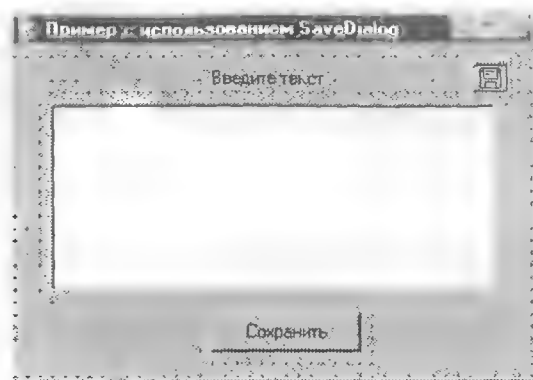


Рис. 13.19. Форма приложения

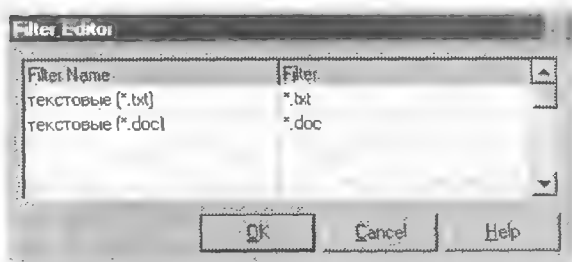


Рис. 13.20. Редактирование фильтров SaveDialog

ПРИМЕЧАНИЕ

Варианты расширения файла и значение расширения по умолчанию можно указать в тексте процедуры записи файла, например, следующим образом:

```
SaveDialog1.Filter:='Text files (*.txt)|*.txt|doc files (*.doc)|*.doc';
SaveDialog1.FilterIndex := 1
```

Для создания процедуры обработчика события щелчка мышью на кнопке Сохранить выберите в окне Инспектора объектов объект Button1 и на странице События произведите двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Введите в раздел описания переменных модуля строковую переменную FName для хранения имени файла:

```
var
  Form1: TForm1;
  FName : string;
```

Отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FName:='Text1';
```

```
SaveDialog1.FileName := FName;      {присвоить свойству FileName
                                     значение из переменной FName}
if SaveDialog1.Execute then begin
  FName:=SaveDialog1.FileName;      {открыть диалог и запомнить новое
                                     имя файла}
  case SaveDialog1.FilterIndex of   {изменить расширение файла}
    FName:=FName+'.txt';
    FName:=FName+'.doc';
  end;
  Memo1.Lines.SaveToFile(FName);    {записать в файл содержимое из
                                     свойства Lines объекта Memo1}
end;
```

Как видно из текста процедуры, сначала переменной FName присваивается значение «Text1», затем это значение присваивается свойству SaveDialog1.FileName. Затем приложение открывает диалоговое окно «Сохранить текстовый файл», в котором задается имя файла. Имя файла из свойства SaveDialog1.FileName запоминается в переменной FName. При помощи оператора case реализуется выбор расширения файла в зависимости от значения свойства SaveDialog1.FilterIndex. В заключительной части процедуры оператор Memo1.Lines.SaveToFile(FName); используется для записи в файл содержимого свойства Lines объекта Memo1.

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Введите текст в окне приложения и щелкните мышью на кнопке Сохранить. Будет открыто диалоговое окно Сохранить текстовый файл, показанное на рис. 13.21.

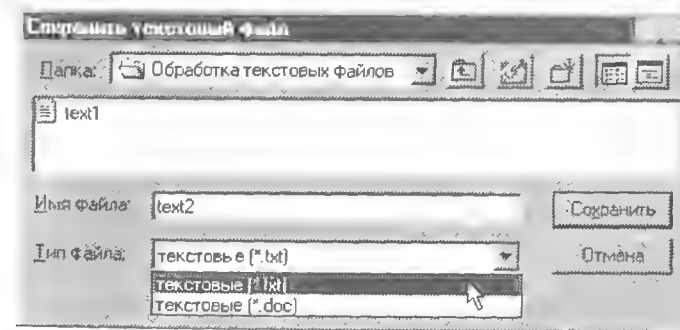


Рис. 13.21. Диалоговое окно сохранения файла

Изменив имя файла и выбрав вариант типа файла, нажмите кнопку Сохранить. Открыв в окне Проводника папку, в которой был сохранен файл (в нашем примере папка называется Обработка текстовых файлов), убедитесь, что в ней присутствует файл с указанным вами именем. Дважды щелкнув мышью по этому файлу, откройте его в окне редактора Блокнот и убедитесь, что

это файл, созданный при проверке работы приложения. Закройте окно редактора Блокнот и приложения Пример с использованием SaveDialog.

Упражнение 12. Создайте приложение, которое открывает текстовый файл с использованием метода OpenFileDialog, считывает текст из него в объект Memo, затем сохраняет измененный текст в файле с использованием метода SaveDialog и выводит текст на печать, используя метод PrintDialog.

Создайте форму и разместите на ней компоненты Memo1, Button1, Button2 и Button3. Задайте для свойства Form1.Caption значение «Пример с OpenFileDialog, SaveDialog, PrintDialog», для Button1.Caption — «Открыть», для Button2.Caption — «Сохранить», для Button3.Caption — «Печатать». Удалите текст «Memo1» из окна Memo1. Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки установите для свойства Memo1.ScrollBars значение ssVertical. Выровняйте и зафиксируйте визуальные компоненты на форме, как показано на рис. 13.22. Выберите в палитре компонентов страницу Dialogs и поместите на форму компоненты OpenFileDialog, SaveDialog и PrintDialog. Так как они не являются визуальными компонентами, то их можно поместить в любое место формы. Задайте для свойства SaveDialog1.Title значение «Сохранить текстовый файл», а для OpenFileDialog1.Title — «Открыть текстовый файл». Эти текстовые строки будут отображаться в заголовках диалоговых окон сохранения и открытия файла.

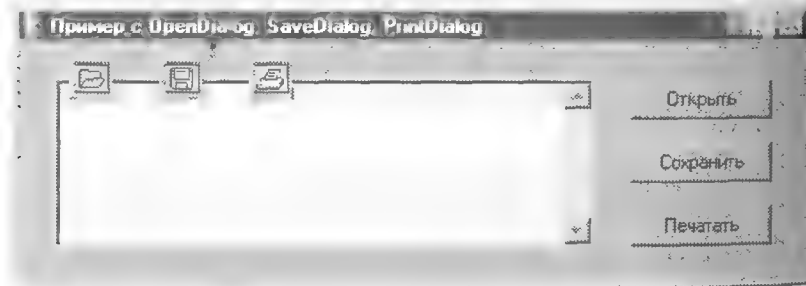


Рис. 13.22. Окно Form1 с компонентами приложения

Чтобы реализовать выбор типа файла при открытии файла в окне диалога, выберите в окне Инспектора объектов объект OpenFileDialog1, на странице Свойства произведите двойной щелчок по списку значений свойства Filter. В окне Filter Editor задайте фильтры для выбора типа и расширения файла, как показано на рис. 13.23.

Чтобы установить в качестве расширения файла по умолчанию первый вариант (*.txt), задайте для свойства OpenFileDialog1.FilterIndex значение 1. Отредактируйте фильтры SaveDialog1, как описано в предыдущем упражнении. Чтобы в диалоговом окне Печать включить возможность выбора диапазона печатаемых страниц и печати выделенного фрагмента, задайте для свойств PrintDialog1.Options.poPageNums и PrintDialog1.Options.poSelection значение True.

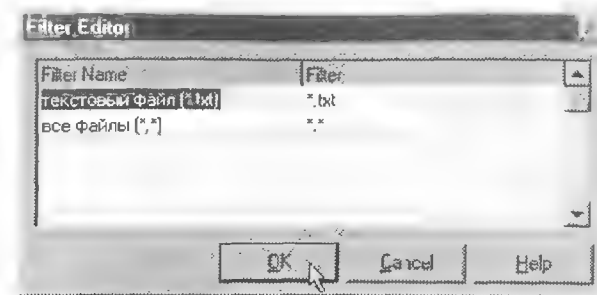


Рис. 13.23. Редактирование фильтров OpenFileDialog

ВНИМАНИЕ

Чтобы можно было использовать принтер, добавьте в список модулей uses модуль PRINTERS. В этом модуле имеется переменная Printer:TPrinter, что избавляет от необходимости описывать свою. Модуль PRINTERS позволяет выводить данные на печать и управлять процессом печати.

После этого раздел описания модулей будет выглядеть следующим образом:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Printers;
```

Отредактируйте раздел описания переменных модуля следующим образом:

```
var
  Form1: TForm1; {описание формы}
  FName : string; {переменная, в которой хранится имя файла}
  F: TextFile; {файловая переменная}
  S: string; {переменная для хранения данных, прочитанных из файла}
```

Процедура обработчика щелчка мышью на кнопке Button2 (Сохранить) создается, как описано в предыдущем упражнении. Чтобы файл можно было сохранять с другим расширением, отредактируйте фрагмент, описывающий выбор типа файла, добавив строки удаления прежнего расширения файла:

```
case SaveDialog1.FilterIndex of
  {изменить расширение файла}
  1 : if Copy(FName, Pos('.', FName), 4) <> '.txt' then begin
        Delete(FName, Pos('.', FName), 4); {удалить расширение файла}
        FName := FName + '.txt';
      end;
  2 : if Copy(FName, Pos('.', FName), 4) <> '.doc' then begin
        Delete(FName, Pos('.', FName), 4);
        FName := FName + '.doc';
      end;
end;
```

Для создания процедуры обработчика щелчка мышью на кнопке Открыть выберите в окне Инспектора объектов объект Button1 и на странице События произведите двойной щелчок на пустом поле списка в событии OnClick.

После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`

Отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject); {открыть файл}
begin
  if OpenFileDialog1.Execute then
  begin
    AssignFile(F, OpenFileDialog1.FileName);      {выбор файла в диалоговом
                                                    окне}
    FName:=OpenDialog1.FileName;                 {имя выбранного файла}
    Reset(F);      {открыть существующий файл}
    Readln(F, S);   {прочитать содержимое файла в переменную S}
    Memo1.Text := S; {присвоить значение S свойству Text объекта Memo1}
    CloseFile(F);
  end;
end;
```

Как видно из текста процедуры, в начале процедуры открывается диалоговое окно, в котором выбирается файл (`FName:=OpenDialog1.FileName;`), затем имя файла связывается с файловой переменной (`AssignFile(F,OpenDialog1.FileName);`), и файл открывается для чтения (`Reset(F);`). После этого содержимое открытого файла считывается в переменную `S` (`Readln(F, S);`) и присваивается свойству `Text` объекта `Memo1` (`Memo1.Text:= S;`). После завершения чтения содержимого файл закрывается (`CloseFile(F);`).

Для создания процедуры обработчика щелчка мышью на кнопке Печатать выберите в окне Инспектора объектов объект `Button3` и на странице События произведите двойной щелчок на пустом поле списка в событии `OnClick`. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button3Click(Sender: TObject);`, которую следует отредактировать следующим образом:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  if PrintDialog1.Execute then
  begin
    AssignPrn(F);      {объявить файловую переменную F - принтер}
    Rewrite(F);         {открыть доступ к принтеру}
    Writeln(F,Memo1.Text); {вывести текст из Memo1 на принтер}
    System.CloseFile(F); {закрыть доступ к принтеру}
  end;
end;
```

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Щелкните мышью на кнопке Открыть, в диалоговом окне Открыть текстовый файл выберите папку, задайте тип файла и выберите текстовый файл, после чего нажмите кнопку Открыть. Отредактируйте текст в окне приложения и нажмите кнопку Сохранить. После этого в диалоговом окне Сохранить текстовый файл выберите в поле Тип файла расширение для сохраняемого файла, задайте

его имя и щелкните мышью на кнопке Сохранить. Для вывода считанного из файла текста на печать щелкните мышью на кнопке Печатать. В диалоговом окне Печать выберите принтер, укажите число копий, диапазон печатаемых страниц и нажмите OK. Открыв в окне Проводника папку, в которой был сохранен файл, убедитесь, что в ней присутствует файл с указанным вами именем. Убедитесь в том, что текст файла напечатан на выбранном вами принтере, и закройте окно приложения.

Контрольные вопросы и задания

Вопросы

1. Опишите 4 типа строк в языке Object Pascal Delphi 5–6: `String`, `ShortString`, `AnsiString`, `WideString`.
2. Опишите назначение процедур и функций обработки строк в стиле Pascal: `Concat`, `Copy`, `Delete`, `FloatToStr`, `Insert`, `Length`, `Pos`, `StrToInt`.
3. Опишите отличия статических и динамических массивов. Каков порядок описания и применения динамических массивов?
4. Опишите назначение операций и функций для массивов: операция присваивания, функции `Length`, `High`, `Low`.
5. Каково назначение и в чем специфика функций для массивов: `MaxIntValue`, `MinIntValue`, `MaxValue`, `MinValue`, `Sum`?
6. Какие функции применимы только к символьным массивам?
7. Каково назначение визуального компонента `GroupBox`? Как задать надпись в этом компоненте?
8. Чем отличаются компоненты `RadioGroup` и `GroupBox`? Как задать список элементов в объекте `RadioGroup`?
9. Для чего и каким образом выполняется фиксация положения компонентов на форме?
10. Каково назначение визуального компонента `StringGrid`? Что содержат его свойства `Align`, `ColCount`, `FixedRows`?
11. С какой целью используется оператор `with..do` для обращения к элементам объекта?
12. Опишите назначение процедур и функций файлового ввода-вывода: `Append`, `AssignFile`, `CloseFile`, `Eof`, `IOResult`, `Read`, `Reset`, `Rewrite`, `Seek`, `Write`.
13. Каково назначение директивы компилятора `{I-}`?
14. Опишите назначение компонентов `OpenDialog`, `SaveDialog`, `PrintDialog` страницы `Dialogs` палитры компонентов. Каковы особенности их размещения на форме приложения?
15. Каково назначение свойства `SaveDialog1.Filter`? Какими способами можно изменять значение этого свойства?

Задания

1. Измените значок приложения, созданного при выполнении упражнения 1, создав, например, значок с изображением буквы «Т» в белом круге.
2. Создайте приложение, предлагающее пользователю ввести строку, а затем определяющее, является ли введенная строка «перевертышем».

ПРИМЕЧАНИЕ

Перевертышем назовем текст, одинаково читающийся слева-направо и справа-налево, например, «шалаш».

3. Создайте приложение, которое предлагает пользователю ввести строку, а затем выводит обращенную строку текста, например, «Привет» — «тевирП».
4. Дополните приложение обработки линейного массива, созданное при выполнении упражнения 3 таким образом, чтобы пользователь мог определить диапазон изменений значений элементов массива.
5. Измените приложение, созданное при выполнении упражнения 8 таким образом, чтобы в создаваемый текстовый файл выводился текст — значение свойства `text` компонента `Мемо`.
6. Создайте приложение, которое выполняет следующие операции с текстовым файлом: открывает текстовый файл с расширением `*.txt` или `*.doc`, указанный пользователем в диалоговом окне, затем изменяет текст, удаляя из него последовательность символов, заключенную в фигурные скобки `{ }` и сохраняет файл на диске с новым именем и расширением, которые задает пользователь в диалоговом окне.

Глава 14

Приложения с мультимедиа

Интегрированная среда программирования Delphi обеспечивает программисту возможность создания приложений, использующих средства мультимедиа.

ПРИМЕЧАНИЕ

Мультимедиа — это интеграция технологий, позволяющая объединить в компьютере практически все виды информационных сообщений: текст, графику, анимацию, аудио- и видеосообщения и обеспечить активное взаимодействие человека с этими данными в реальном масштабе времени.

Рассмотрим примеры использования графики, анимации, аудио и видео в приложениях, созданных в среде Delphi.

Канва и пикселы

Многие компоненты в Delphi имеют свойство `Canvas` (канва, холст), представляющее собой область компонента, на которой можно рисовать или отображать готовые изображения. Свойство `Canvas` типа `TCanvas` используется для рисования пером `Pen` и кистью `Brush` для модификации изображения и наложения друг на друга нескольких изображений. Класс `TCanvas` является основой графической подсистемы Delphi. Канва обеспечивает:

- загрузку и хранение графических изображений;
- создание новых и изменение имеющихся изображений с помощью пера, кисти, шрифта;
- рисование и закраску различных фигур, линий, текстов;
- комбинирование различных изображений

Канва содержит свойства и методы, существенно упрощающие работу с графикой в Delphi. С ними можно подробно ознакомиться при помощи справочной системы Delphi. Каждая точка канвы имеет координаты `X` и `Y`. Система координат канвы, как и везде в Delphi, имеет началом левый верхний угол канвы. Координата `X` возрастает при перемещении слева направо, а координата `Y` — при перемещении сверху вниз. Координаты измеряются в пикселах. Пиксел — это наименьший элемент поверхности рисунка, с которым можно

манипулировать. Важнейшее свойство пиксела — его цвет. Для описания цвета используется тип TColor. Рисовать на канве можно разными способами, например, можно рисовать, окрашивая пиксеты канвы.

Рисование на канве по пикселям

Для рисования на канве пикселями используется свойство канвы Pixels. Это свойство представляет собой двумерный массив, который отвечает за цвета канвы. Например, Canvas.Pixels[15,25] соответствует цвету пиксела 15-го слева и 25-го сверху. С массивом пикселей можно обращаться как с любым свойством: изменять цвет, присваивать пикселу новое значение или определять его цвет по хранящемуся в нем значению. Например, при помощи операторов Canvas.Pixels[15,25] := 0 или Canvas.Pixels[15,25] := clBlack для пиксела задается черный цвет.

Упражнение 1. Создайте приложение, которое строит пикселями график функции $y=3*\sin(x)*\cos(x/6)$.

Создайте форму, свойству Caption которой присвойте значение «График функции». В окне Инспектора объектов задайте для свойства Form1.ClientHeight (высота окна формы) значение 300, для свойства Form1.ClientWidth (ширина окна формы) задайте значение 500, для свойства Form1.Color задайте значение clWhite (белый).

Для создания процедуры обработчика события OnPaint (перерисовать изображение) выберите в окне Инспектора объектов объект Form1 и на странице Events произведите двойной щелчок на пустом поле списка в событии OnPaint. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.FormPaint(Sender: TObject);

Введите в текст процедуры вызов процедуры построения графика функции DrawGraph:

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    DrawGraph; {вызов процедуры построения графика функции}
end;
```

Так как при программировании процедуры построения графика функции нам придется использовать для рисования Canvas (холст, канва), окрашивая пиксеты канвы, изучите справочную информацию о свойствах Canvas и Pixels. Поскольку перед построением графика придется чертить прямые линии — оси координат и выводить текст на изображении, то следует также изучить справку Delphi о методах MoveTo, LineTo и TextOut.

В связи с тем, что при построении графика функции необходимо задавать целочисленные значения координат окрашиваемого пиксела, а результат вычисления функции является вещественным числом, следует ознакомиться со справочной информацией Delphi, касающейся функции Round, округляющей вещественные числа до ближайшего целого числа.

Откройте окно Редактора кода и введите текст процедуры построения графика функции:

```
procedure DrawGraph; // Построение графика функции
```

```
var
    x1, x2, y1, y2, dx, l, b, h, w, x0, y0: real;
    mY, mX: integer;
    {границы изменения аргумента функции}
    {границы изменения значения функции}
    {аргумент функции}
    {значение функции в точке x}
    {приращение аргумента}
    {левый нижний угол области вывода графика}
    {ширина и высота области вывода графика}
    {масштаб по осям X и Y}
    {точка - начало координат}

function f(x: real): real {функция, график которой надо построить}
begin
    f := 3 * Sin(x) * Cos(x / 6);
end;

begin
    {область вывода графика на форме}
    {X - координата левого верхнего угла}
    {Y - координата левого верхнего угла}
    l := 10;
    b := Form1.ClientHeight - 20;
    h := Form1.ClientHeight - 40; // высота
    w := Form1.Width - 40; // ширина
    x1 := 0; // нижняя граница диапазона аргумента
    x2 := 25; // верхняя граница диапазона аргумента
    dx := 0.01; // шаг аргумента

    {определение максимального и минимального
    значений функции на отрезке [x1,x2]
    для масштабирования}
    y1 := f(x1); {минимальное значение функции}
    y2 := f(x2); {максимальное значение функции}

    {вычислять значения функции для значений
    x <= x2 и определить максимальное и
    минимальное значения для масштабирования}

    repeat
        y := f(x);
        if y < y1 then y1 := y;
        if y > y2 then y2 := y;
        x := x + dx;
    until x >= x2;
    mY := h * abs(y2 - y1); // масштаб по оси Y
    mX := w * abs(x2 - x1); // масштаб по оси X
    x0 := 1;
```

```

y0:=b-Abs(Round(y1*my));
with Form1.Canvas do
begin
    {рисование графика}
    {начертить оси координат}
    MoveTo(1,b): LineTo(1,b-h); {ось x}
    MoveTo(x0,y0): LineTo(x0+w,y0); {ось y}
    TextOut(1+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
    {вывести максимальное значение}
    TextOut(x0+w,y0,'x'); // подписать ось x
    TextOut(1+5,b,FloatToStrF(y1,ffGeneral,6,3));
    {вывести минимальное значение}
    TextOut(1-7,b-h-5,'y'); // подписать ось y
    {построение графика}
    x:=x1;
    repeat {повторять рисовать точки}
        y:=f(x);
        Form1.Canvas.Pixels[x0+Round(x*mx),y0-Round(y*my)]:=clRed;
        {clRed - красный цвет рисования графика}
        x:=x+dx;
    until (x>=x2); {прекратить, как только x>=x2}
end;
end;

```

Изучите текст процедуры построения графика функции DrawGraph. Сохраните, откомпилируйте и запустите приложение на выполнение. Убедитесь, что в окне приложения производится построение графика заданной функции, как показано на рис. 14.1.

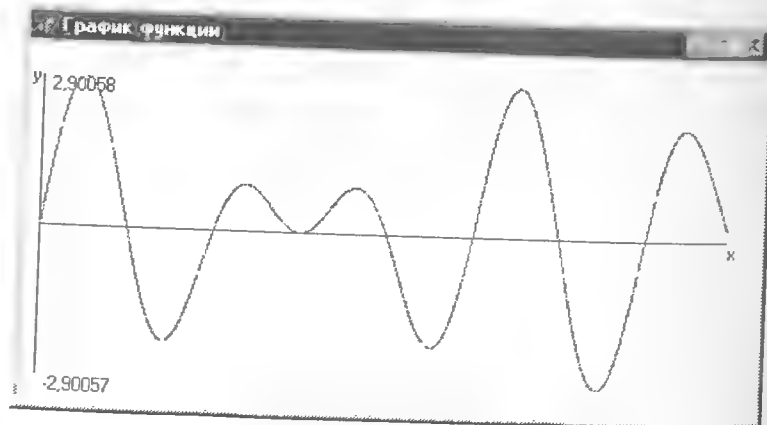


Рис. 14.1. Окно приложения с графиком функции

Если вы попытаетесь изменить размеры окна, то увидите, что график искажается, так как он не перерисовывается в соответствии с новыми размерами окна.

Для того чтобы перерисовывать изображение при изменении размеров формы, следует создать обработчик события OnResize, определяющий новые параметры клиентской области и вызывающий процедуру построения графика функции DrawGraph. Эта процедура может выглядеть следующим образом:

```

procedure TForm1.FormResize(Sender: TObject);
begin
    with Form1 do
        Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));
        {заполнение прямоугольника канвы, указанного
        параметром Rect, где ClientWidth.ClientHeight -
        ширина и высота клиентской области в пикселах}

        DrawGraph;
    end;
end;

```

Сохраните измененное приложение, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения выполняется построение графика заданной функции, причем при изменении размеров окна происходит корректное перерисовывание графика в новом масштабе.

Упражнение 2. Измените приложение таким образом, чтобы оно выполняло построение графика функции $y = \sqrt{x} \cdot \sin(x)$, причем цвет графика должен быть синим.

Для изменения выражения функции отредактируйте текст функции f(x) следующим образом:

```

Function f(x:real):real; {функция, график которой надо построить}
begin
    f:=Sqrt(x)*Sin(x)
end;

```

Для изменения цвета графика измените значение свойства Form1.Canvas.Pixels с красного (clRed) на синий (clBlue) в тексте фрагмента процедуры DrawGraph, рисующего точки на канве:

```

repeat {рисовать точки}
    y:=f(x);
    Form1.Canvas.Pixels[x0+Round(x*mx),
    y0-Round(y*my)]:=clBlue; {clBlue - синий цвет}
    x:=x+dx;
until (x>=x2); {прекратить, как только x>=x2}

```

Сохраните измененное приложение, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения выполняется построение графика заданной функции, причем цвет пикселей графика — синий.

Рисование пером

У канвы имеется свойство Pen (перо), которое определяет атрибуты пера, используемого для рисования линий и фигур. Свойство Pen, в свою очередь, имеет ряд подсвойств. Одно из них — уже известное вам свойство Color—

представляет цвет, которым наносится рисунок. Второе свойство — *Width* (ширина линии), которая задается в пикселах. По умолчанию ширина равна 1. Свойство *Style* определяет вид линии (сплошная, штриховая, пунктирная и т. п.). Все стили со штрихами и пунктирами доступны только при *Width = 1*, в противном случае линии этих стилей рисуются как сплошные.

Для определения в координатах канвы текущей позиции пера используется свойство канвы *PenPos* типа *TPoint*. Перемещение пера без прорисовки линии, т. е. изменение *PenPos*, производится при помощи метода канвы *MoveTo(X,Y)*. Здесь *X* и *Y* — координаты точки, в которую перемещается перо. Эта текущая точка становится исходной, от которой при помощи метода *LineTo(X,Y)* можно провести линию в точку с координатами (*X,Y*). При этом текущая точка перемещается в конечную точку линии и новый вызов *LineTo* будет проводить линию из этой новой текущей точки.

Упражнение 3. Создайте приложение, которое по щелчку мышью на кнопке выполняет рисование пером графика функции $y = \sin(x) \cdot \cos(x/6)$ на интервале $X_{\min} = 0$, $X_{\max} = 4\pi$. Задайте толщину пера 2 пиксела, цвет линии *clBlue* (синий).

Создайте форму и задайте для ее свойства *Caption* значение «Рисование графика функции пером». Поместите в верхнюю часть формы компонент *Image1* из палитры *Additional*. Ниже *Image1* на форме разместите компонент *Button1* из палитры *Standard* и задайте для его свойства *Caption* значение «Построить график».

Рисование графика функции пером опишите в процедуре обработчика щелчка мышью на кнопке *Button1*. Для этого, выбрав в окне Инспектора объектов *Button1*, на странице *Events* произведите двойной щелчок на пустом поле списка в событии *OnClick*.

Так как в процедуре обработчика события необходимо обрабатывать свойства *Pen*, *Font*, а также использовать методы *LineTo*, *MoveTo*, *TextOut*, вам следует изучить соответствующие разделы справочной системы *Delphi*.

После этого в окне Редактора кода отредактируйте текст процедуры обработчика события *TForm1.Button1Click* следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X,Y : real;           {значения аргумента и функции}
  PX,PY : longint;      {координаты пикселей, соответствующих X,Y}
begin
  with Image1.Canvas do
  begin
    Pen.Width:=1;        {задать толщину пера для рисования осей
                          координат}
    Pen.Color:=clBlack;  {задать черный цвет линии}
                        {начертить оси координат: начало координат
                          в точке 5,Image1.Height div 2}
    MoveTo(0,Image1.Height div 2);
```

```
    LineTo(Image1.Width,Image1.Height div 2); {ось x}
    MoveTo(5,0);
    LineTo(5,Image1.Height);                 {ось y}
                                          {рисовать график от начала координат
                                           на всю ширину Image1}
    Pen.Width:=2;          {задать толщину пера для рисования графика}
    Pen.Color:=clBlue;     {задать синий цвет линии}
    MoveTo(5,Image1.Height div 2); {установить перо в начало координат}
    for PX:=5 to Image1.Width do
    begin
      X:=PX*4*Pi/Image1.Width; {вычисление значения аргумента X
                                для масштабирования графика
                                в соответствии с шириной Image1}
      Y:=Sin(X)*Cos(X/6);       {вычисление значения функции}
      PY:=Trunc(Image1.Height-(Y+1)*Image1.Height/2);
                                {вычисление координаты пиксела,
                                 соответствующей значению функции Y}
      LineTo(PX,PY);           {рисование линии}
    end;
    Font.Size:=18;             {задать размер шрифта в пунктах}
    TextOut(180,20,'y=Sin(x)*Cos(x/6)');    {подписать график}
  end;
end;
```

Изучите текст процедуры и комментарии. Сохраните текст приложения, откомпилируйте его и проверьте его работу. Щелкнув мышью на кнопке *Построить график*, убедитесь, что в окне приложения выполняется рисование графика заданной функции, причем цвет пикселей — синий, а толщина линии составляет 2 пиксела, как показано на рис. 14.2.

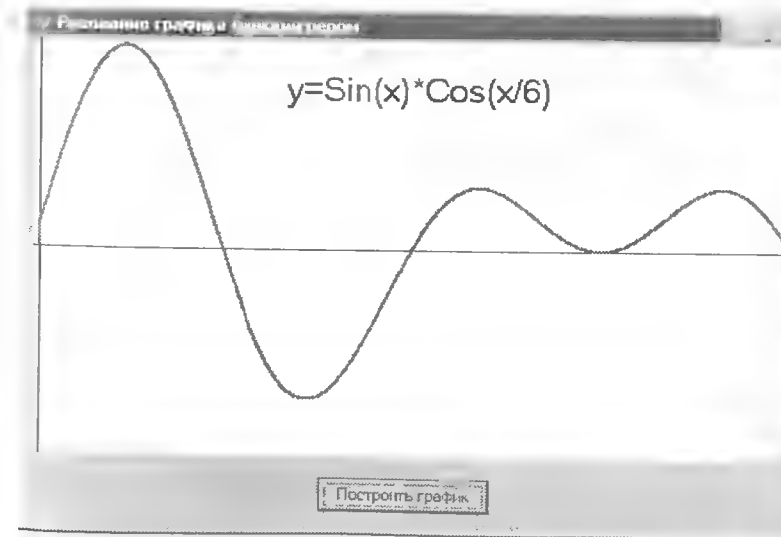


Рис. 14.2. Рисование графика функции пером

Перо может рисовать не только прямые линии, но и геометрические фигуры. Для рисования фигур пером используются следующие методы:

Метод	Действие
Arc	Дуга окружности или эллипса
Chord	Замкнутая фигура, ограниченная дугой окружности (или эллипса) и хордой
Ellipse	Окружность или эллипс
Pie	Сектор окружности или эллипса
PolyBezier	Кусочная кривая третьего порядка с точным отображением первой и последней точки; число точек должно быть кратно 3
PolyBezierTo	Кусочная кривая третьего порядка с точным отображением последней точки; число точек должно быть на 1 меньше числа, кратного 3
Polygon	Замкнутая фигура с кусочно-линейной границей
Polyline	Кусочно-линейная кривая
Rectangle	Прямоугольник
RoundRect	Прямоугольник со скругленными углами

```
Polyline([Point(100,100),Point(160,50),Point(220,100)]): {крыша}
Ellipse(140,70, 180,90): {окошко на чердаке}
Rectangle(115,130,150,200): {дверь}
Rectangle(165,130,205,165): {окно}
Font.Size:=16: {задать размер шрифта в пунктах}
TextOut(140,220,'Домик'): {подписать рисунок}
end:
```

Изучите текст процедуры и комментарии. Сохраните текст приложения, откомпилируйте его и проверьте его работу. Щелкнув мышью на кнопке Нарисовать, убедитесь, что в окне приложения выполняется рисование домика, как показано на рис. 14.3.

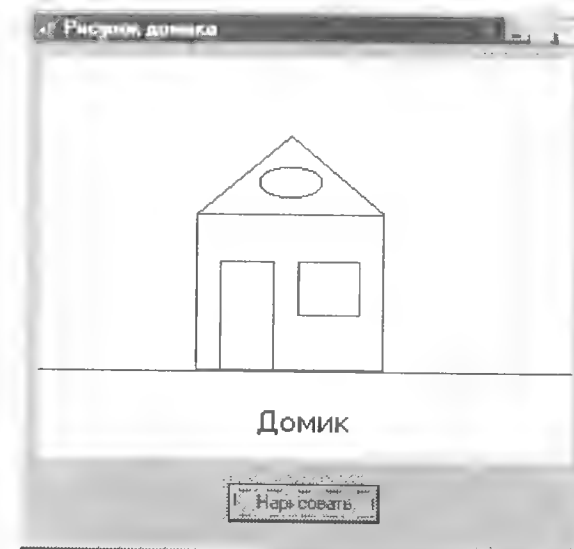


Рис. 14.3. Окно приложения с нарисованным домиком

Рисование кистью

У канвы имеется свойство **Brush** — кисть. Оно определяет фон и заполнение замкнутых фигур на канве. Свойство **Brush** имеет, в свою очередь, ряд под-свойств: **Color**, **Bitmap**, **Handle** и **Style**.

Color — цвет кисти. По умолчанию — **clWhite** (белый).

Handle — дескриптор кисти окна, определяющий доступ к дескриптору объекта GDI Windows.

Style — определяет шаблон заполнения (штриховку).

Bitmap — указатель на внешнюю матрицу растрового изображения, используемую в качестве шаблона заполнения. Свойство кисти — **Bitmap** — определяет нестандартное заполнение заданным шаблоном, который задается матрицей

Упражнение 4. Создайте приложение, которое рисует пером домик.

Создайте форму и задайте для ее свойства **Caption** значение «Рисунок домика». Поместите в верхнюю часть формы компонент **Image1** из палитры **Additional**. Ниже **Image1** на форме разместите компонент **Button1** из палитры **Standard** и задайте для его свойства **Caption** значение «Нарисовать».

Рисование пером графика функции опишите в процедуре обработчика щелчка мышью на кнопке **Button1**. Для этого, выбрав в окне Инспектора объектов **Button1**, на странице **Events** произведите двойной щелчок на пустом поле списка в событии **OnClick**.

Так как в процедуре обработчика события необходимо обрабатывать свойства **Pen**, **Font**, а также использовать методы **LineTo**, **MoveTo**, **TextOut**, **Arc**, **Chord**, **Ellipse**, **Pie**, **PolyBezier**, **PolyBezierTo**, **Polygon**, **Polyline**, **Rectangle**, **RoundRect**, вам следует ознакомиться с соответствующими разделами справочной системы **Delphi**.

После этого в окне Редактора кода отредактируйте текст процедуры обработчика события **TForm1.Button1Click** следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1.Canvas do
  begin
    Pen.Width:=1;           {задать толщину пера для рисования}
    Pen.Color:=clBlack;     {задать черный цвет линии}
    MoveTo(0,200);
    LineTo(Image1.Width,200); {начертить поверхность земли}
    MoveTo(100,200);
    Rectangle(100,100,220,200); {стены}
    MoveTo(100,200);
```


размером 8 на 8 разрядов. Если для кисти задан шаблон Bitmap, то заполнение производится именно этим шаблоном, независимо от значения свойства Style. Шаблон Bitmap может создаваться в процессе выполнения приложения или, например, загружаться из файла, как в приведенном ниже примере:

```
var
  Bitmap: TBitmap;
begin
  Bitmap := TBitmap.Create;           {создание объекта}
  try
    Bitmap.LoadFromFile('MyBitmap.bmp'); {загрузка в объект Bitmap
                                          матрицы из файла}

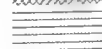
    Image1.Canvas.Brush.Bitmap := Bitmap;

                                          {использование шаблона для
                                          заполнения фигур на канве
                                          Image1}

  finally
    Image1.Canvas.Brush.Bitmap := nil; {обнуление указателя}
    Bitmap.Free;                       {освобождение памяти}
  end;
end;
```

В этом примере создается объект Bitmap типа TBitmap и в него загружается матрица из файла с именем «MyBitmap.bmp». Затем свойству Image1.Canvas.Brush.Bitmap присваивается указатель на этот объект. После этого загруженный шаблон можно использовать для заполнения фигур на канве Image1. В конце фрагмента кода свойству Bitmap присваивается значение nil, после чего заполнение опять начинает определяться свойством Style. Затем объект Bitmap уничтожается, чтобы освободить занимаемую им память.

Свойство Style может принимать следующие значения:

Значение	Шаблон	Значение	Шаблон
bsSolid		bsCross	
bsClear		bsDiagCross	
bsBDiagonal		bsHorizontal	
bsFDiagonal		bsVertical	

Например, следующие операторы изменяют цвет и стиль заполнения объекта Image1.Canvas — канвы компонента Image1:

```
Image1.Canvas.Brush.Style := bsCross;
Image1.Canvas.Brush.Color := clRed;
Image1.Canvas.FillRect(Rect(0,0,Image1.Width,Image1.Height));
```

Последний из приведенных операторов заполняет при помощи метода FillRect всю поверхность канвы. Метод FillRect позволяет выполнить рисование заполненных фигур. Он объявляется как:

```
procedure FillRect(const Rect: TRect);
```

Он позволяет заполнить заданным стилем или шаблоном прямоугольную область, заданную параметром Rect. Этот параметр имеет тип TRect. Для его задания проще всего использовать функцию Rect(X1,Y1,X2,Y2), возвращающую структуру Rect с координатами углов, заданных параметрами (X1, Y1) и (X2, Y2).

Функцию FillRect удобно использовать для стирания изображения. Например, оператор

```
with Image1 do Canvas.FillRect(Rect(0,0,Width,Height));
```

очищает всю площадь канвы компонента Image1.

Кисть участвует в заполнении фигур не только с помощью этой функции. Все перечисленные ранее методы рисования замкнутых фигур могут использоваться в сочетании с кистью. Это относится к методам Chord, Ellipse, Pie, Polygon и к остальным методам, рассмотренным при изучении рисования пером.

При работе с кистью также используется метод FloodFill, который заполняет замкнутую область на канве. Этот метод определен следующим образом:

```
type TFillStyle = (fsSurface, fsBorder);
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

Точка с координатами X и Y является произвольной внутренней точкой заполняемой области, которая может иметь произвольную форму. Граница этой области определяется сочетанием параметров Color и FillStyle. Параметр Color указывает цвет, который используется при определении границы заполняемой области, а параметр FillStyle определяет, как именно по этому цвету определяется граница. Если FillStyle = fsSurface, то заполняется область, окрашенная цветом Color, а на других цветах кисть останавливается. Если FillStyle = fsBorder, то, наоборот, заполняется область, окрашенная любыми цветами, не равными Color, а на цвете Color кисть останавливается.

Для определения области закрашивания можно использовать координаты и цвет одного из пикселей, расположенных внутри области (если FillStyle = fsSurface) или снаружи ее (если FillStyle = fsBorder). Например, приведенные ниже операторы закрашивают белым цветом на канве компонента Image1 все пиксели, прилегающие к пикселу с координатами (X, Y) и имеющие тот же цвет, что и этот пиксел.

```
with Image1.Canvas do
begin
  Brush.Color:=clWhite;
  FloodFill(X,Y,Pixels[X,Y].fsSurface);
end;
```

Приведенные ниже операторы закрашивают белым цветом на канве компонента `Image1` все пиксели, прилегающие к пикселу с координатами (X, Y) и имеющие цвет, отличный от черного. При достижении черной границы области закрашка прекращается.

```
with Image1.Canvas do
begin
  Brush.Color:=clWhite;
  FloodFill(X,Y,clBlack, fsBorder);
end;
```

Имеется еще один метод канвы, связанный с кистью. Это метод `FrameRect`. Он рисует на канве текущей кистью прямоугольную рамку, не закрашивая ее. Синтаксис метода `FrameRect`: `procedure FrameRect(const Rect: TRect);` Параметр `Rect` определяет позицию и размеры прямоугольной рамки. Толщина рамки составляет 1 пиксел. Область внутри рамки не заполняется. Метод `FrameRect` отличается от рассмотренного ранее метода `Rectangle` тем, что рамка рисуется цветом кисти (в методе `Rectangle` — цветом пера), и область не закрашивается (в методе `Rectangle` она закрашивается).

Например, оператор

```
with Image1.Canvas do
begin
  Brush.Color := clBlack;
  FrameRect(Rect(10,10,100,100));
end;
```

рисует на канве компонента `Image1` черную рамку.

При работе с кистью имеется возможность использовать метод `TextRect` для более красивого вывода текста. Например, чтобы нарисовать в заданном месте канвы компонента `Image1` красный прямоугольник и внутри него по центру вывести текст, введенный пользователем в окно редактирования `Edit1`, можно создать следующий код:

```
var st:string;
    X1,Y1,X2,Y2:integer;
begin
  st:=Edit1.Text;
  X1:=100; Y1:=100; X2:=200; Y2:=150; {координаты прямоугольника}
  with Image1.Canvas do
  begin
    Brush.Color:=clRed;           {задать красный цвет кисти}
                                {вывести текст, выровненным
                                по центру прямоугольника}

    TextRect (Rect(X1,Y1,X2,Y2).
              X1+(X2-X1-TextWidth(st)) div 2,
              Y1+(Y2-Y1-TextHeight(st)) div 2,st);

  end;
end;
```

ПРИМЕЧАНИЕ

Если текст оказывается длиннее ширины прямоугольника, то он усекается. В данном примере будет видна только середина текста, так как текст выровнен по центру.

Если в приведенном примере заменить оператор `TextRect` на

```
TextRect(Rect(X1-5,Y1-5, X1+TextWidth(st)+5,
              1+TextHeight(st)+5), X1, Y1, st);
```

то текст будет выводиться полностью в красной прямоугольной области, на 5 пикселей отступающей во все стороны от текста.

Упражнение 5. Создайте простой графический редактор, который дает пользователю возможность загрузить рисунок, используя команду меню **Файл ▶ Открыть**, выбрать цвет и рисовать кистью, а также отменить операцию редактирования при помощи команды **Правка ▶ Отмена**.

Создайте форму и задайте для ее свойства `Caption` значение «Графический редактор». Поместите в нижнюю левую часть формы два компонента типа `TImage`, придайте им квадратную форму, например, 17 на 17. Это будут окна основного и вспомогательного цветов. Присвойте этим компонентам имена `Image1` и `Image2`.

Поместите в верхнюю правую часть формы еще один компонент `Image3` и задайте его размеры таким образом, чтобы он занимал основную часть формы, оставив поле в левой части формы место для инструментальной панели размером, например, 240 на 240 пунктов. Это будет холст для рисования. Присвойте этому компоненту имя `Image3`.

Перенесите на форму еще один компонент типа `TImage` и расположите его в нижней части формы правее первых двух на одном с ними уровне. Это будет палитра цветов. Сделайте ее высоту такой же, как и у первых двух компонентов (17 пунктов), а длину — примерно в 10 раз большую (160 пунктов). Присвойте этому компоненту имя `Image4`.

Разместите в верхнем левом углу формы кнопку типа `TSpeedButton`. Эта кнопка будет соответствовать кисти — типичному инструменту графических редакторов. В окне Инспектора объектов выберите объект `SpeedButton1` и присвойте его свойству `Name` значение `SbBrush`. Чтобы обеспечить кнопке возможность фиксироваться в нажатом и ненажатом состоянии, установите у кнопки `SbBrush` для свойства `GroupIndex` значение 1, а для свойства `AllowAllUp` — значение `True`. Чтобы отобразить на кнопке рисунок кисти, загрузите в свойство `Glyph` значок кисти. Для этого в окне Инспектора объектов выберите объект `SbBrush` и на странице `Properties` произведите двойной щелчок мышью в поле значения свойства `Glyph`, затем в окне `Picture Editor` щелкните мышью на кнопке `Load` и в окне `Load Picture` выберите соответствующий файл формата `*.bmp` и щелкните мышью на кнопке `Открыть`, как показано на рис. 14.4. Щелкнув мышью на кнопке `ОК` в окне `Picture Editor`, завершите выбор графического файла для отображения на кнопке.

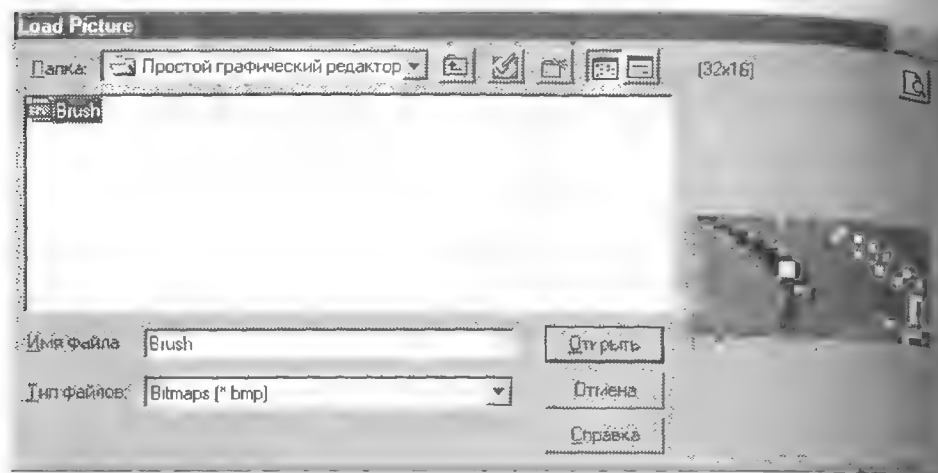


Рис. 14.4. Выбор графического файла для отображения на кнопке

Поместите на форму ниже кнопки SbBrush еще одну кнопку типа TSpeedButton и назовите ее SbColor. Установите для свойства SbColor.GroupIndex значение 1 (это гарантирует, что только одна из кнопок SbBrush или SbColor может быть нажата), а для свойства AllowAllUp — значение True. Загрузите в свойство Glyph значок (например, файл one2one.bmp).

Чтобы обеспечить использование диалога открытия файла, поместите на форму диалог OpenPictureDialog.

ПРИМЕЧАНИЕ

Если вы работаете с Delphi 1–2, то вам следует поместить компонент OpenFileDialog и задать для его свойства Filter значение (bitmaps |*.bmp| все файлы|*.*).

Поместите на форму компонент типа TMainMenu. Свойства и методы компонента MainMenu обеспечивают объединение меню главной и вспомогательной форм и связь с меню OLE-контейнера.

Свойство Items содержит массив разделов меню типа TMenuItem, обладающих своими свойствами, методами и событиями. Свойство Caption обозначает заголовки раздела, свойство Name — имя объекта раздела, свойство Shortcut определяет клавиши оперативного доступа к разделу. Свойство Default определяет, является ли данный раздел разделом по умолчанию для своего подменю, т. е. разделом, выполняемым при двойном щелчке мышью по родительскому разделу. Свойство Break используется в длинных меню, чтобы разбить список разделов на несколько столбцов. Свойство Checked, установленное в True, указывает, что в разделе меню будет отображаться маркер флажка, показывающий, что данный раздел выбран. Еще одним свойством, позволяющим вводить маркеры в разделы меню, является RadioItem. Это свойство, будучи установленным в True, определяет, что данный раздел должен работать в режи-

ме переключателя совместно с другими разделами, имеющими то же значение свойства GroupIndex. Свойства Enabled (доступен) и Visible (видимый) могут быть установлены для каждого из разделов во время проектирования или выполнения программы.

Основное событие раздела меню — OnClick, возникающее при щелчке мышью по разделу или при нажатии клавиш оперативного доступа.

ПРИМЕЧАНИЕ

Начиная с версии 4, в Delphi предусмотрена возможность ввода в разделы меню изображений. Для этого предназначены свойства разделов Bitmap и ImageIndex. Первое из них позволяет непосредственно ввести изображение в раздел, выбрав его из файла. Второе позволяет указать индекс изображения, хранящегося во внешнем компоненте TImageList. Указание на этот компонент можно задать в свойстве Images компонента TMainMenu.

Команды контекстного меню конструктора Create Submenu позволяют добавить в выделенный раздел подменю. Например, чтобы вставить подменю, нужно указать позицию и в контекстном меню выбрать команду Insert, как показано на рис. 14.5.

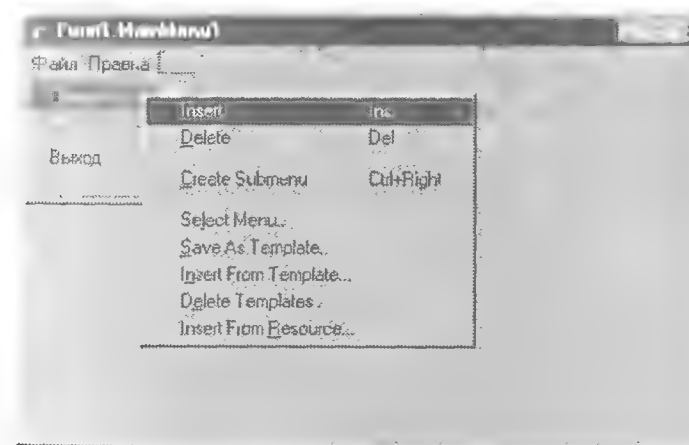


Рис. 14.5. Конструирование меню приложения

Используя конструктор меню, создайте меню приложения, показанное на рис. 14.5.

ПРИМЕЧАНИЕ

Для ввода разделителя между пунктами подменю вставьте этот пункт командой Insert контекстного меню и задайте для его свойства Caption значение «-».

В меню создайте раздел Файл с подразделами Открыть и Выход. Конструктор меню присваивает пунктам меню по умолчанию имена N1, N2, N3 и N4. Чтобы имена пунктов меню в текстах процедур обработчиков событий были понятнее,

измените их названия, изменив значения свойств Name соответствующих объектов. При этом нельзя пользоваться зарезервированными словами Object Pascal. Например, задайте для свойств Name пунктов меню значения «OpenFile» и «Exit». Для разделителя пунктов меню N3 свойство Name изменять не нужно, так оно не будет использоваться. Создайте в главном меню еще один раздел — Правка с подразделом Отменить и назовите этот подраздел «EditUndo». После этого состав меню будет представлен в окне Object TreeView, как показано на рис. 14.6.



Рис. 14.6. Состав меню простого графического редактора

В результате размещения компонентов и конструирования меню получится форма приложения примерно следующего вида (рис. 14.7).

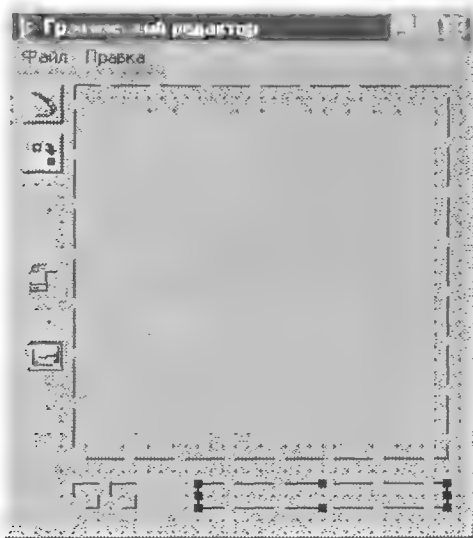


Рис. 14.7. Вид формы приложения «Графический редактор»

Сохраните файлы программного модуля и проекта в папке «Простой графический редактор». Для обеспечения работы приложения отредактируйте

исходный текст модуля, добавив в него описания переменных и процедур-обработчиков событий.

Чтобы сгенерировать процедуру обработчика события при создании формы приложения, выберите в окне Инспектора объектов объект Form1 и на странице Events произведите двойной щелчок на пустом поле списка в событии OnCreate. Добавьте в раздел implementation объявление переменной

```
var Bitmap: TBitmap;
```

В этой переменной будет сохраняться изображение, чтобы его можно было восстановить командой Правка ► Отменить.

Отредактируйте текст процедуры обработчика следующим образом:

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
  HW: integer;           {HW - ширина отдельной ячейки в палитре цветов,  
                        1 - номер ячейки в палитре цветов}
```

```
begin
```

```
  Bitmap:=TBitmap.Create;
```

```
  Image1.Canvas.Brush.Color:=clBlack; {задать основной цвет кисти}
```

```
  Image2.Canvas.Brush.Color:=clWhite; {задать вспомогательный цвет кисти}  
                                     {заполнение окон основного  
                                     и вспомогательного цветов}
```

```
  with Image1.Canvas do FillRect(Rect(0,0,Width,Height));
```

```
  with Image2.Canvas do FillRect(Rect(0,0,Width,Height));
```

```
  HW:=Image4.Width div 10;           {задание ширины ячейки палитры  
                                     цветов}
```

```
                                     {закраска элементов палитры цветов}
```

```
  with Image4.Canvas do
```

```
    for i:=1 to 10 do
```

```
      begin
```

```
        case i of {в зависимости от i задать цвет}
```

```
          1 · Brush.Color:=clBlack; {черный}
```

```
          2 · Brush.Color:=clAqua; {голубой}
```

```
          3 · Brush.Color:=clBlue; {синий}
```

```
          4 · Brush.Color:=clFuchsia; {сиреневый}
```

```
          5 · Brush.Color:=clGreen; {зеленый}
```

```
          6 · Brush.Color:=clLime; {лимонно-зеленый}
```

```
          7 · Brush.Color:=clMaroon; {темно-бордовый}
```

```
          8 · Brush.Color:=clRed; {красный}
```

```
          9 · Brush.Color:=clYellow; {желтый}
```

```
         10 · Brush.Color:=clWhite; {белый}
```

```
        end;
```

```
        Rectangle((i-1)*HW,0,i*HW,Height); {заполнить выбранным цветом  
                                             прямоугольник-ячейку палитры  
                                             цветов}
```

```
      end;
```



```

{рисование изображения креста
на холсте — простейшего рисунка
в окне редактора}

with Image3 do begin
  Canvas.MoveTo(0,0):
    {переместить перо в левый верхний
    угол Image3}
  Canvas.LineTo(Width,Height):
    {провести линию в правый нижний
    угол Image3}
  Canvas.MoveTo(0,Height):
    {переместить перо в левый нижний
    угол Image3}
  Canvas.LineTo(Width, 0):
    {провести линию в правый верхний
    угол Image3}
end;
Bitmap.Assign(Image3.Picture);
end;
```

Как видно из текста процедуры, сначала создается объект **Bitmap**, в котором будет храниться первоначальное изображение. Затем задаются свойства кисти окон основного (**Image1**) и вспомогательного (**Image2**) цветов: черный и белый. Окна закрашиваются соответствующим цветом с помощью функции **FillRect**. После этого формируется палитра цветов — для каждого элемента палитры задается свой цвет и элемент закрашивается этим цветом с помощью функции **Rectangle**. Затем на холсте **Image3** рисуется крест для демонстрации простого изображения. В конце процедуры нарисованное на канве изображение сохраняется в объекте **Bitmap** методом **Assign**.

Для завершения работы приложения создайте обработчик события формы **OnDestroy** и запишите в теле процедуры обработчика оператор **Bitmap.Free**, который освобождает память при закрытии приложения.

```

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Bitmap.Free; {освобождение памяти}
end;
```

Чтобы можно было загрузить изображение из файла, создайте обработчик выбора команды **Файл ▶ Открыть**, для чего в окне Инспектора объектов выберите объект **FileOpen** и на странице **Events** произведите двойной щелчок на пустом поле списка в событии **OnClick**. Отредактируйте текст процедуры следующим образом:

```

procedure TForm1.FileOpenClick(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
  begin
    Image3.Picture.LoadFromFile(OpenPictureDialog1.FileName);
    Bitmap.Assign(Image3.Picture);
  end;
```

Как видно из текста процедуры, сначала пользователь с помощью **OpenPictureDialog1** выбирает файл изображения, а затем это изображение запоминается в **Bitmap**.

Чтобы можно было в процессе редактирования выполнить отмену операции редактирования, создайте обработчик события выбора команды **Правка ▶ Отменить** для чего в окне Инспектора объектов выберите объект **EditUndo** и на странице **Events** произведите двойной щелчок на пустом поле списка в событии **OnClick** и вставьте в текст процедуры оператор **Image3.Picture.Assign(Bitmap)**, который восстанавливает на холсте изображение, которое ранее было сохранено в **Bitmap**. Текст процедуры обработчика отмены операции редактирования будет выглядеть следующим образом:

```

procedure TForm1.EditUndoClick(Sender: TObject);
begin
  Image3.Picture.Assign(Bitmap);
end;
```

Чтобы перед началом работы с очередным инструментом запомнить текущий вид изображения, создайте обработчик события **OnClick** для кнопок **SbBrush** и **SbColor**, а в текст процедуры добавьте оператор

```

if (Sender as TSpeedButton).Down then
  Bitmap.Assign(Image3.Picture);
```

Создайте обработчик события **OnMouseDown** для компонентов **Image3** и **Image4** и отредактируйте текст процедуры обработчика следующим образом:

```

procedure TForm1.Image3MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if (Sender=Image4) or SbColor.Down then
    {режим установки основного и
    вспомогательного цветов}

  if Button=mbLeft then
    with Image1.Canvas do
      {установка основного цвета}
      Brush.Color:=(Sender as TImage).Canvas.Pixels[X,Y];
      FillRect(Rect(0,0,Width,Height));

    with Image2.Canvas do
      {установка вспомогательного цвета}
      Brush.Color:=(Sender as TImage).Canvas.Pixels[X,Y];
      FillRect(Rect(0,0,Width,Height));
    end;
  else if SbBrush.Down then
    with Image3.Canvas do
```

```

begin
    {режим закраски указанной области холста}
    if Button=mbLeft then
        Brush.Color:=Image1.Canvas.Brush.Color
    else Brush.Color:=Image2.Canvas.Brush.Color;
    FloodFill(X,Y.Pixels[X,Y].fsSurface);
end;
end;

```

Эта процедура производит установку основного и вспомогательного цветов, а также выполняет функцию инструмента графического редактора — кисти. Если левая кнопка мыши нажата на палитре цветов Image4 или если выбрана кнопка SbColor (кнопка указателя цвета), то приложение находится в режиме установки цветов. При нажатой левой кнопке мыши цвет пиксела под курсором мыши передается в окно основного цвета, а при нажатии правой кнопки — в окно вспомогательного цвета.

Если кнопка мыши нажата на холсте Image3 и при этом выбрана кнопка SbBrush, то приложение находится в режиме закраски указанной области рисунка. В этом случае, в зависимости от нажатой кнопки мыши, выбирается основной или вспомогательный цвет и функцией FloodFill производится закрашка области, координаты внутренней точки которой указаны курсором мыши, а цвет — цветом пиксела, на который указывает курсор мыши.

Создайте процедуру обработки выбора команды меню **Файл** ▶ **Выход**, для чего в окне Инспектора объектов выберите объект Exit, на странице Events произведите двойной щелчок на пустом поле списка в событии OnClick и вставьте в текст процедуры оператор Close, который закрывает окно приложения. Текст процедуры будет выглядеть следующим образом:

```

procedure TForm1.ExitClick(Sender: TObject);
begin
    Close; {закрывает окно приложения}
end;

```

Измените значок приложения. Для этого выберите в меню Delphi команду **Project** ▶ **Options**, в окне Project Options откройте вкладку Application и щелкните мышью на кнопке Load Icon. В окне Application Icon выберите папку и файл значка, например, Paint, и щелкните мышью на кнопке Открыть, как показано на рис. 14.8.

Щелкнув на кнопке ОК в окне Project Options, завершите изменение значка приложения. Сохраните текст приложения, откомпилируйте его и выполните. Проверьте установку цветов. При щелчке левой или правой кнопкой мыши по палитре цветов будет меняться, соответственно, основной или вспомогательный цвет. Если выбрать кисть, то при щелчке на холсте указанная курсором область должна закрашиваться основным или вспомогательным цветом, как показано на рис. 14.9, а. Если отпустить кнопку кисти в окне приложения и нажать кнопку определения цветов, то при щелчке мышью по холсту основной

или вспомогательный цвет будет устанавливаться равным цвету холста под курсором мыши. Для загрузки изображения из файла выберите в меню редактора команду **Файл** ▶ **Открыть**, затем в диалоговом окне Открытие файла выберите папку и файл изображения и щелкните мышью на кнопке Открыть. На рис. 14.9, б показано окно приложения с загруженным из файла изображением.

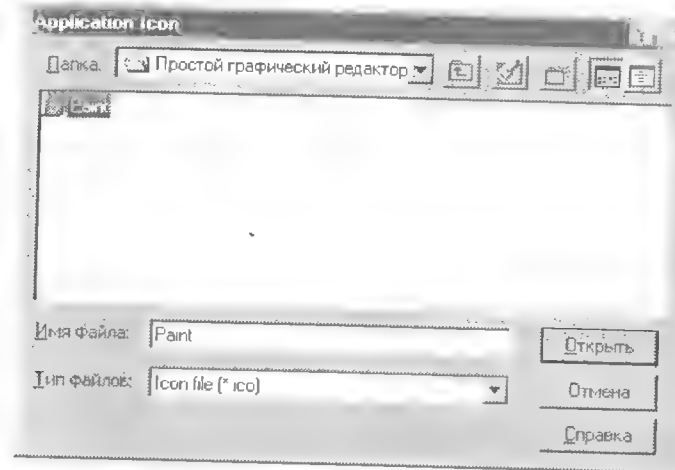


Рис. 14.8. Выбор файла значка

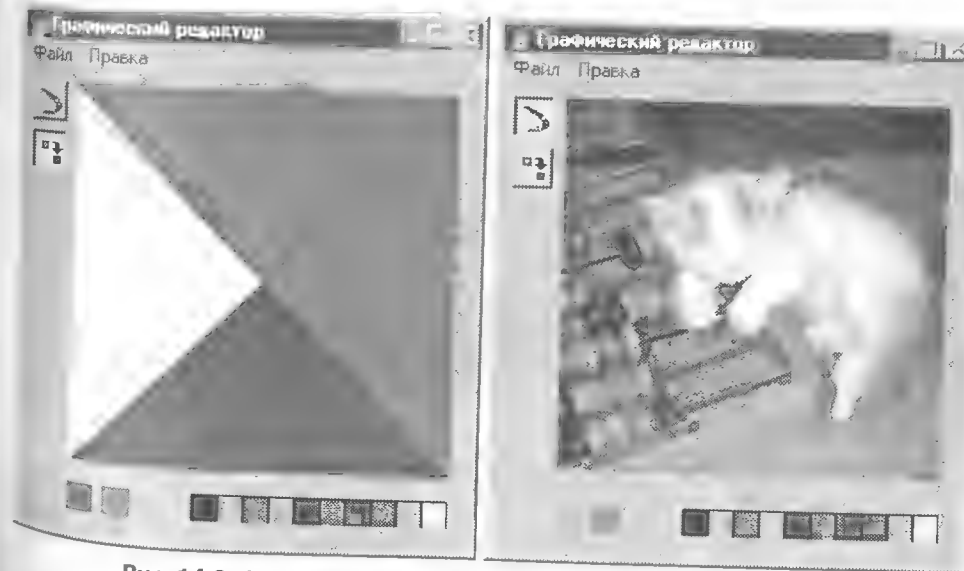


Рис. 14.9. Окно графического редактора: а) с тестовым рисунком; б) с изображением, загруженным из файла

Поэкспериментируйте с кистью и цветами. Если потребуется отказаться от изменений, внесенных в изображение, выберите команду **Правка** ▶ **Отменить**.



Рис. 14.12. Редактирование значка приложения в файле ресурсов

Для запуска анимации создайте обработчик события `OnActivate`, наступающего, когда форма получает или теряет фокус, например, при щелчке по ней мышью. Чтобы создать обработчик события, выберите в окне Инспектора объектов объект `Form1` и на странице `Events` произведите двойной щелчок на пустом поле списка в событии `OnActivate`. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.FormActivate(Sender: TObject);`

Отредактируйте текст этой процедуры следующим образом:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
    Back := TBitmap.Create;    // фон
    bitmap := TBitmap.Create;  // картинка
    Buf := TBitmap.Create;     // буфер
                                {загрузить и вывести фон из файла
                                Background.bmp}
    Back.LoadFromFile('Background.bmp');
    Form1.Image1.Canvas.Draw(0,0,Back);
                                {загрузить изображение самолета,
                                которое будет двигаться}
    bitmap.LoadFromFile('airplane.bmp');
    bitmap.Transparent := true;
    bitmap.TransparentColor := bitmap.Canvas.Pixels[1,1];
```

{создать буфер для сохранения копии области фона, на которую накладывается картинка}

```
w:= bitmap.Width;
h:= bitmap.Height;
Buf.Width:= W;
Buf.Height:=H;
Buf.Palette:=Back.Palette; {обеспечить соответствие палитр}
Buf.Canvas.CopyMode:=cmSrcCopy;
BufRct:=Bounds(0,0,W,H);
x:= -w
y:=20
BackRct:=Bounds(x,y,W,H); {определить сохраняемую область фона}
Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct); {сохранить ее}
end;
```

Для получения эффекта мультипликации создайте процедуру обработчика события `OnTimer`, для чего выберите в окне Инспектора объектов объект `Timer1` и на странице `Events` произведите двойной щелчок на пустом поле списка в событии `Timer1Timer`. Отредактируйте тело процедуры обработчика события следующим образом:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Form1.Image1.Canvas.Draw(x,y,Buf); {восстановлением фона (из буфера) -
                                         удалить рисунок}
    x:=x+2;                             {смещение самолета по оси x}
    if x>Form1.Image1.Width then x:=-W; {если самолет "вылетел" за пределы
                                         рисунка фона}
    BackRct:=Bounds(x,y,W,H);           {определить сохраняемую область
                                         фона}
    Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct); {сохранить ее копию}
    Form1.Image1.Canvas.Draw(x,y,bitmap); {вывести рисунок
                                         самолета в новой
                                         позиции}
end;
```

Для освобождения памяти от переменных, в которых хранились значения фона, рисунка самолета и буфера с фрагментом фона, создайте процедуру обработчика события закрытия окна приложения. Для этого выберите в окне Инспектора объектов объект `Form1` и на странице `Events` произведите двойной щелчок на пустом поле списка в событии `OnClose`. Вставьте в тело процедуры обработчика события вызов метода `Free` для освобождения памяти, динамически выделенной под объекты `Back`, `bitmap` и `Buf`.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Back.Free;    {освободить память}
    bitmap.Free;
    Buf.Free;
end;
```

Сохраните текст приложения, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения на фоне выбранного рисунка осуществляется перемещение изображения самолета, как показано на рис. 14.13.

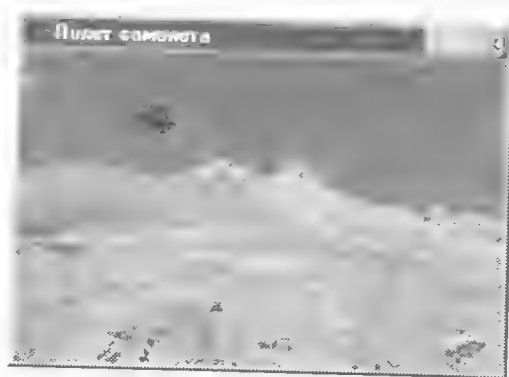


Рис. 14.13. Окно приложения с самолетом, летящим на фоне изображения

Воспроизведение звука и видеоклипов

Так как мультимедиа-технологии позволяют объединить графику, анимацию, аудио и видео, вкратце рассмотрим аудио- и мультимедиа-файлы. Наиболее простым аудиофайлом является волновой файл формата .wav. В нем записывается цифровое представление информации о волновой форме электрического сигнала, соответствующего каждому звуку. Wav-файлы имеют большой размер, так как волновой файл «не знает» ничего о том, что такое звук и что он означает. из-за чего для хранения звукового клипа в файле формата .wav приходится запоминать большое количество информации. Другим часто применяемым типом файлов-носителей являются файлы цифрового интерфейса музыкальных инструментов (MIDI). Файлы .midi используются для хранения музыкальных фрагментов. В этих файлах звук хранится в виде данных о том, на каких инструментах исполняются определенные ноты и как долго они звучат. Отсюда и главный недостаток файлов MIDI — на разном оборудовании они звучат по-разному. Одним из главных преимуществ MIDI является то, что файлы получаются сравнительно небольшими. Трекерная музыка (MOD, STM, S3M, XM, IT) применялась первоначально для музыкального сопровождения компьютерных игр. Файлы этого формата отличаются тем, что помимо нотной партитуры, в файлы включаются еще и инструменты в оцифрованном виде. При воспроизведении компьютер в реальном масштабе времени микширует их и выдает на звуковую плату результирующий сигнал. Таким образом, в отличие от MIDI, эти файлы звучат одинаково практически на любом компьютере. Кроме того, в них могут содержаться оригинально звучащие инструменты и даже вокал. Инструменты могут быть получены путем оцифровки звукового сигнала или синтезирова-

ны и записаны в WAV-формате, который легко преобразуется в формат инструмента трекера (PAT, SMP).

Волновые и MIDI-файлы могут хранить только звук или музыку. Для хранения видеoinформации разработан ряд других форматов. Отметим среди них файлы AVI и MPEG. Большинство видеофайлов поддерживает также хранение звуковой дорожки, так что звук воспроизводится синхронно с изображением. В основу хранения видеoinформации заложено представление о том, что человеческий мозг интерпретирует быструю последовательность изображений, незначительно отличающихся друг от друга, как движение. Каждое из таких изображений называется *кадром*. Каждый следующий кадр несколько отличен от предыдущего. Чтобы мозг воспринимал смену кадров как плавное движение, желательно воспроизводить не менее 24 кадров в секунду. Более высокая частота не приводит к заметному росту качества, а более низкая производит впечатление мерцания экрана. Если бы каждый кадр хранился в файле в виде растрового изображения (а это несколько сотен килобайт), то потребовался бы огромный объем дисковой памяти. При такой простой схеме хранения на компакт-диск, например, можно было бы записать всего 72 секунды видеофильма. Поэтому на практике для хранения видеофильмов используется сжатие видеоданных.

Процедуры воспроизведения звуков

Наиболее простой процедурой, управляющей звуком, является процедура Beep. Она не имеет параметров и воспроизводит стандартный звуковой сигнал, установленный в Windows, если компьютер имеет звуковую карту и задан стандартный сигнал (он устанавливается в Панели управления Windows после щелчка мышью по значку Звук). Если звуковой карты нет, или стандартный сигнал не установлен, звук воспроизводится через динамик компьютера в виде короткого щелчка.

Более серьезной функцией является MessageBeep. Она определена как

```
Function MessageBeep(uType:word):boolean;
```

Параметр uType указывает воспроизводимый звук как идентификатор раздела [sounds] реестра, в котором записаны звуки, сопровождающие те или иные события в Windows. С помощью приложения Звук в Панели управления Windows пользователь может удалить или установить соответствующие звуки. Для параметра uType определены следующие константы:

Значение	Звук
MB_ICONASTERISK	Звездочка
MB_ICONEXCLAMATION	Восклицание
MB_ICONHAND	Критическая ошибка
MB_ICONQUESTION	Вопрос
MB_OK	Стандартный звук

После инициализации воспроизведения звука функция `MessageBeep` возвращает управление в точку вызова и воспроизведение звука производится асинхронно. Если функция `MessageBeep` не находит указанный тип звука, она пытается воспроизвести стандартный звук. Если и он не установлен или если компьютер не снабжен звуковой картой, то звук воспроизводится через динамик компьютера.

При успешном завершении функция возвращает ненулевое значение (`true`). Если функция вернула нулевое значение, то получить информацию об ошибке можно с помощью вызова `GetLastError`.

Упражнение 8. Создайте приложение, которое позволит прослушать стандартные звуки Windows. Для выбора воспроизводимого звука используйте переключатели, размещенные в панели `RadioGroup`.

Создайте форму и задайте для ее свойства `Caption` значение «Тестирование звука». Поместите на форму компоненты `Button` и `RadioGroup`, как показано на рис. 14.8. Задайте для свойства `Caption` кнопки `Button` значение «Послушать». Задайте для свойства `Caption` компонента `RadioGroup` значение «Вариант звучания». Так как количество переключателей в группе и подписей к ним определяется свойством `Items`, выберите в Инспекторе объектов компонент `RadioGroup1` и на странице `Events` выберите свойство `Items` (список элементов). В окне `String List Editor` введите список элементов: `Beep`, `Звездочка`, `Восклицание`, `Критическая ошибка`, `Вопрос`, `Стандартный звук`. Щелкнув мышью на кнопке `OK`, завершите формирование списка вариантов прослушиваемых звуков Windows. Выровняйте и зафиксируйте компоненты на форме. После этого форма с размещенными на ней компонентами должна выглядеть, как показано на рис. 14.14.

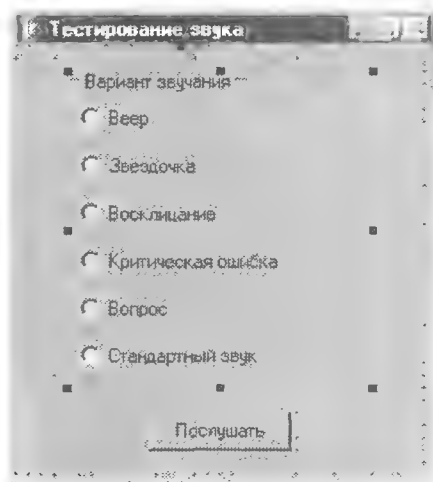


Рис. 14.14. Форма приложения для прослушивания стандартных звуков Windows

Для создания процедуры обработки события — щелчка мышью на кнопке `Послушать`, в окне Инспектора объектов выберите объект `Button1`, затем на странице `Events` произведите двойной щелчок на пустом поле списка в событии `OnClick`. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject)`.

Так как в процедуре обработки нажатия кнопки `Button1` должно быть шесть вариантов реализации для разных звуков, то рационально записать выбор варианта звука с помощью оператора `case`. Текст процедуры обработчика события может быть записан следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  case RadioGroup1.ItemIndex of
    0 : Beep; {выбор варианта звука}
    1 : MessageBeep(MB_ICONASTERISK); {стандартный звуковой сигнал}
    2 : MessageBeep(MB_ICONEXCLAMATION); {звездочка}
    3 : MessageBeep(MB_ICONHAND); {восклицание}
    4 : MessageBeep(MB_ICONQUESTION); {критическая ошибка}
    5 : MessageBeep(MB_OK); {вопрос}
  end; {стандартный звук Windows}
end;
```

Сохраните файлы проекта и модуля в папке «Стандартные звуки», а затем откомпилируйте и запустите приложение.

ПРИМЕЧАНИЕ

Для успешной демонстрации работы приложения необходимо, чтобы на компьютере были установлены звуки, соответствующие различным типам сигналов Windows. Для их установки выберите команду `Пуск` ▶ `Настройка` ▶ `Панель управления`, а затем, дважды щелкнув мышью по значку `Звук`, откройте диалоговое окно `Свойства: Звук`.

В окне приложения `Тестирование звука`, выбирая варианты звучания с помощью переключателей и щелкая мышью на кнопке `Послушать`, убедитесь в правильной работе приложения.

Функция `PlaySound`

Функция `PlaySound` воспроизводит указанный волновой файл, или звук системного события, или звук из ресурса. Функция `PlaySound` определена следующим образом:

```
function PlaySound(pszSound:PChar; hmod:HINST;
  fdwSound:Cardinal):boolean;
```

Параметр `pszSound` представляет собой строку с нулевым символом в конце и определяет воспроизводимый звук. В зависимости от значения флага `fdwSound` (`SND_FILENAME`, `SND_ALIAS` или `SND_RESOURCE`), параметр `pszSound`

может определять имя волнового файла, псевдоним системного события или идентификатор ресурса. Если ни один из этих флагов не указан, функция ищет в реестре Windows или в файле WIN.INI указанное имя звука. Если звук найден, то он воспроизводится. Если звук не найден, то параметр pszSound интерпретируется как имя файла.

Если параметр pszSound равен 0, то воспроизведение любого волнового файла прерывается. Для прерывания воспроизведения звука, не связанного с волновым файлом, следует указать SND_PURGE в параметре fdwSound.

Параметр hmod используется только при параметре fdwSound, равном SND_RESOURCE. В этом случае hmod является дескриптором исполняемого файла, содержащего ресурс, который должен загружаться. В противном случае значение hmod задается равным 0.

Параметр fdwSound задает флаги воспроизведения звука. Флаги могут комбинироваться друг с другом при помощи операции or. Наиболее важные для воспроизведения значения параметра fdwSound:

Параметр	Назначение
SND_ASYNC	Звук воспроизводится асинхронно и функция PlaySound возвращает управление немедленно после начала воспроизведения. Чтобы прекратить асинхронное воспроизведение волнового файла, следует вызвать PlaySound с параметром pszSound, равным 0
SND_FILENAME	Параметр pszSound является именем файла
SND_LOOP	Воспроизведение звука постоянно повторяется, пока не будет вызвана функция PlaySound с параметром pszSound, равным 0. Одновременно должен быть указан флаг SND_ASYNC асинхронного воспроизведения звука
SND_NOSTOP	Если заданный звук не может быть воспроизведен, поскольку ресурсы, необходимые для воспроизведения, заняты воспроизведением другого звука, функция PlaySound немедленно возвращает false, не воспроизводя заданный звук. Если этот флаг не указан, функция PlaySound пытается остановить воспроизведение другого звука, чтобы устройство могло быть использовано для воспроизведения нового звука
SND_NOWAIT	Если драйвер занят, функция немедленно возвращает управление без воспроизведения заданного звука
SND_PURGE	Останавливает воспроизведение любых звуков, вызванных в данной задаче. Если pszSound не равен 0, прекращается воспроизведение всех экземпляров указанного звука. Если pszSound равен 0, то прекращается воспроизведение всех звуков, связанных с данной задачей. Отдельно следует указать дескриптор для остановки событий SND_RESOURCE
SND_RESOURCE	Параметр pszSound является идентификатором ресурса. Параметр hmod должен указывать на источник ресурса
SND_SYNC	Синхронное воспроизведение звука события. Функция PlaySound возвращает управление только после окончания воспроизведения

ВНИМАНИЕ

Функция PlaySound — функция API Windows, параметры которой описаны в модуле mmsystem, поэтому для использования этой функции необходимо включить в раздел uses ссылку на mmsystem.

Звук, указанный параметром pszSound, должен помещаться в доступную память и должен подходить для установленного драйвера устройства воспроизведения волновых файлов. Функция PlaySound ищет файл звука в следующих каталогах: текущем, каталоге Windows, системном каталоге Windows, каталогах, перечисленных в переменной среды PATH, в списке каталогов, предоставляемых сетью.

Если указанный звук не найден, то функция PlaySound воспроизводит системный звук по умолчанию. Если функция не может найти и его, то воспроизведения не производится, а функция возвращает значение false.

Упражнение 9. Создайте приложение, которое позволит прослушивать музыкальные файлы, открывая их с помощью OpenFileDialog.

Создайте форму и задайте для ее свойства Caption значение «Прослушивание звуковых файлов». Поместите в любое место формы компонент OpenFileDialog из палитры Dialogs и задайте для свойства OpenFileDialog1.Title значение «Открыть звуковой файл», которое будет отображаться в заголовке диалогового окна открытия файла. Задайте для свойства OpenFileDialog1.Filter значение «звуковые файлы (*.wav) |*.wav». Поместите на форму компонент Button и задайте для его свойства Caption значение «Прослушать». Для создания обработчика щелчка мышью на кнопке Button1 в окне Инспектора объектов выберите объект Button1, затем на странице Events произведите двойной щелчок на пустом поле списка в событии OnClick, после этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.Button1Click(Sender: TObject).

Так как в процедуре обработки нажатия кнопки Button1 используется вызов функции PlaySound, то следует включить в раздел описания Uses вызов модуля mmSystem. Использование OpenFileDialog для открытия звукового файла реализуется при помощи оператора

```
if OpenFileDialog1.Execute then
  PlaySound(PChar(OpenDialog1.FileName).0, SND_ASYNC);
```

где параметр SND_ASYNC указывает на асинхронное воспроизведение звука.

Сохраните файлы проекта и модуля в папке «Прослушивание звуковых файлов». Откомпилируйте и запустите приложение. Щелчком мышью на кнопке Послушать в окне приложения откройте диалоговое окно Открыть звуковой файл, в котором выберите звуковой файл, например в папке C:\Windows\Media, как показано на рис. 14.15.

Прослушав звук, закройте окно приложения.

Функция PlaySound позволяет воспроизводить и системные звуки, просто называя их псевдонимы. Например, оператор PlaySound('SystemStart', 0, SND_ASYNC); воспроизведет тот же звук, что и приведенный ранее оператор, указывавший


```

Animate1.CommonAVI:=aviFindFolder; {воспроизвести первый стандартный
                                   клип - поиск в папке}
Label1.Caption:='Клип Windows: поиск в папке';
Animate1.Active:=True;             {активизировать анимацию}
end;

```

Создайте процедуру обработчика щелчка мышью на кнопке BtStop и отредактируйте ее текст следующим образом:

```

procedure TForm1.BtStopClick(Sender: TObject);
begin
  Animate1.Stop; {вызов процедуры смены стандартного видеоклипа Windows}
end;

```

Процедуру смены стандартного видеоклипа Windows реализуйте в виде процедуры обработчика события окончания воспроизведения видеоклипа OnStop. Для этого в окне Инспектора объектов выберите компонент Animate1 и на странице Events произведите двойной щелчок на пустом поле списка в событии OnStop. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.Animate1Stop(Sender: TObject);. Отредактируйте ее текст следующим образом:

```

procedure TForm1.Animate1Stop(Sender: TObject);
begin
  Inc(I); {увеличить счетчик клипов}
  with Animate1 do begin
    case I of
      2: begin
          CommonAVI:=aviFindFile;
          Label1.Caption:='Клип Windows: поиск файла';
        end;
      3: begin
          CommonAVI:=aviFindComputer;
          Label1.Caption:='Клип Windows: поиск компьютера';
        end;
      4: begin
          CommonAVI:=aviCopyFiles;
          Label1.Caption:='Клип Windows: копирование файлов';
        end;
      5: begin
          CommonAVI:=aviCopyFile;
          Label1.Caption:='Клип Windows: копирование файла';
        end;
      6: begin
          CommonAVI:=aviRecycleFile;
          Label1.Caption:='Клип Windows: удаление файла в корзину';
        end;
      7: begin
          CommonAVI:=aviEmptyRecycle;
          Label1.Caption:='Клип Windows: очистка корзины';
        end;
    end;
  end;
end;

```

```

B: begin
  CommonAVI:=aviDeleteFile;
  Label1.Caption:='Клип Windows: удаление файла';
end;
end;
if I<9 then Active:=True
else
  begin
    Visible:=False;
    Label1.Caption:=''. {очистить после показа клипов}
  end;
end;
end;
end;

```

Как видно из текста процедуры, выбор вида стандартного клипа описан с помощью оператора выбора case. Причем значение селектора I изменяется от 2 до 8, так как для I = 1 воспроизведение видеоклипа выполняется в процедуре обработчика щелчка мышью на кнопке BtWind. Каждый оператор, выполняемый для конкретного значения селектора I, является составным, например,

```

begin
  CommonAVI:=aviFindFile;
  Label1.Caption:='Клип Windows: поиск файла';
end;

```

где CommonAVI:=aviFindFile; определяет конкретный стандартный видеоклип Windows, а затем Label1.Caption:='Клип Windows: поиск файла'; выводит в окне приложения сообщение о том, какой клип воспроизводится.

Для воспроизведения видеоклипов из файлов формата *.avi создайте процедуру обработчика щелчка мышью на кнопке BtFile. Отредактируйте текст этой процедуры следующим образом:

```

procedure TForm1.BtFileClick(Sender: TObject);
var
  FName: string; {переменная для хранения части имени файла}
begin
  if OpenDialog1.Execute then
    with Animate1 do
      begin
        I:=9;
        FileName:=OpenDialog1.FileName;
        FName:=FileName;
        repeat {удалить часть имени до '\'}
          Delete(FName,1,Pos('\',FName));
        until Pos('\',FName)=0;
        Visible:=True;
        Label1.Caption:='Клип из файла '+FName;
        Active:=True;
      end;
    end;
end;

```

Как видно из текста процедуры, в разделе описания переменных введена переменная FName, в которую копируется значение из FileName, с тем чтобы впоследствии «отрезать» от полного имени файла имя папки, в которой он находится. Для открытия файла видеоклипа используется стандартный диалог OpenFileDialog1. Фрагмент программы:

```
repeat {удалить все в имени до ' '}
  Delete(FName, 1, Pos(' ', FName), FName)
until Pos(' ', FName)=0
```

выполняет удаление из полного имени файла имени папки, в которой он находится.

Сохраните файлы проекта и модуля в папке «Воспроизведение немых видеоклипов». Откомпилируйте и запустите приложение. Щелкнув мышью на кнопке Клипы Windows в окне приложения, просмотрите воспроизведение стандартного видеоклипа. Для завершения просмотра текущего стандартного клипа Windows и перехода к следующему используйте щелчок мышью на кнопке Стоп. Просмотрите таким образом все клипы Windows. Щелкнув мышью на кнопке Клип из файла, в диалоговом окне Открыть файл видеоклипа выберите файл видеоклипа и щелкните мышью на кнопке Открыть для просмотра видеоклипа, как показано на рис. 14.17.



Рис. 14.17. Окно приложения с видеоклипом

Для остановки проигрывания видеоклипа щелкните мышью на кнопке Стоп. Закройте окно приложения.

Универсальный проигрыватель аудио- и видеоинформации MediaPlayer

Для проигрывания аудио и видеоинформации в Delphi (начиная с Delphi 2) имеется универсальный проигрыватель аудио- и видеофайлов — MediaPlayer. Этот компонент по умолчанию входит в состав палитры System и инкапсулирует интерфейс управления носителями (Media Control Interface, MCI)

Windows 95 и Windows NT. MediaPlayer имеет ряд кнопок, управляющих такими устройствами мультимедиа, как CD-ROM и др., управляемых мышью. Общий вид компонента представлен на рис. 14.18.



Рис. 14.18. Вид компонента MediaPlayer

Назначение кнопок MediaPlayer (слева направо):

Кнопка	Значение	Действие
Play	btPlay	Воспроизведение
Pause	btPause	Пауза воспроизведения или записи (если проигрыватель в момент щелчка уже находится в состоянии паузы, то воспроизведение или запись возобновляется)
Stop	btStop	Останов воспроизведения или записи
Next	btNext	Переход на следующий трек или на конец записи
Prev	btPrev	Переход на предыдущий трек или на начало записи
Step	btStep	Перемещение вперед на заданное число кадров
Back	btBack	Перемещение назад на заданное число кадров
Record	btRecord	Начало записи
Eject	btEject	Освобождение объекта, загруженного в устройство

Проигрыватель может управляться как кнопками, так и непосредственно при помощи соответствующих этим кнопкам методов (Play, Pause, Stop, Next, Previous, Step, Back, StartRecording, Eject). В этом случае сам компонент TMediaPlayer может быть сделан невидимым.

Тип устройства мультимедиа (dtWaveAudio или dtVideodisc) определяется свойством DeviceType. Если устройство хранит объект воспроизведения в файле, то имя файла задается свойством FileName. Если свойство DeviceType задано равным dtAutoSelect, то проигрыватель пытается определить тип устройства, исходя из расширения имени файла FileName. Если для свойства AutoOpen установлено значение true, то проигрыватель пытается открыть устройство, указанное свойством DeviceType, автоматически во время своего создания в процессе выполнения приложения.

Контрольные вопросы и задания

Вопросы

1. Что такое мультимедиа?
2. Каково назначение свойства Canvas?
3. В чем разница между рисованием на канве по пикселам и пером?
4. Каково назначение методов MoveTo, LineTo, TextOut?

5. Как задать координаты и цвет пиксела канвы?
6. Как задается толщина, цвет пера и стиль линии, которую оно рисует?
7. Каково назначение свойства канвы Brush (кисть)? Какие подсвойства имеются у кисти?
8. Опишите назначение методов FillRect, FloodFill, FrameRect.
9. Каким образом можно задать изображение на кнопке?
10. Опишите назначение компонента MainMenu и порядок использования конструктора меню.
11. Что такое анимация? Какими способами можно создать в приложении Delphi анимированное изображение?
12. Каково отличие звуковых файлов форматов *.wav и *.midi?
13. Сравните возможности функций Beep, MessageBeep и PlaySound.
14. Сравните назначение и возможности компонентов Animate и MediaPlayer.

Задания

1. Измените приложение, созданное при выполнении упражнения 1, таким образом, чтобы выполнялось построение графика функции $y = 2 * \sin x \times \exp(x / 4)$.
2. Создайте приложение, которое при щелчке на кнопке Нарисовать рисует в окне приложения цветок ромашки.
3. Создайте приложение, в окне которого отображается анимация — движение мяча, брошенного под углом к горизонту.
4. Создайте приложение-мультипликацию, в котором на фоне дороги перемещается автомобиль (вариант: на фоне звездного неба перемещается космический корабль).
5. Создайте приложение, которое позволит прослушивать звуковые файлы формата *.wav, открывая их с помощью диалогового окна Открыть звуковой файл, вызываемого командой Файл ► Открыть. Для завершения работы приложения используйте команду Выход в главном меню.

Глава 15

Создание простых приложений

Консольное приложение

Наряду с приложениями, предоставляющими пользователю стандартный оконный интерфейс, Delphi позволяет создавать консольные приложения — приложения для DOS, в которых не используются средства графического интерфейса пользователя.

Упражнение 1. Создайте консольное приложение, которое предлагает ввести два целых числа, затем вычисляет их сумму и выводит на экран результат вычисления. Алгоритм решения этой задачи был разобран в главе 1.

Создать консольное приложение можно несколькими способами.

1. Удалив форму из проекта и работая непосредственно с головным файлом программы.
 - Выберите команду Project ► Options... и в окне Project Options на странице Linker установите опцию Generate Console Application.
 - Или включите в программу директиву компилятора {\$APPTYPE CONSOLE}.
2. Выбрав команду File ► New ► Other, а затем в окне New Items на вкладке New выбрав значок Console Application, как показано на рис. 15.1, и щелкнув мышью на кнопке OK.

После этого в окне интегрированной среды Delphi будет открыто окно проекта с шаблоном программы:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
begin
  { TODO -c:\User -cConsole Main . Insert code here }
end.

```

Как видно из текста программы, в него уже вставлена директива компилятора {\$APPTYPE CONSOLE}, указывающая на генерацию консольного приложения.



Рис. 15.1. Создание консольного приложения

ВНИМАНИЕ

Для правильного отображения русских шрифтов следует использовать функцию CharToOem из модуля Windows, которая выполняет преобразование текста из кодировки Windows1251 (ANSI) в кодировку 866 (OEM).

Чтобы использовать функцию CharToOem, вставьте в раздел описания Uses модуль Windows. В раздел описания переменных поместите объявления целых переменных A, B и Summ, которые будут хранить значения слагаемых и суммы, а также переменной типа PChar — указатель на массив элементов типа Char (строка текста может быть представлена как массив символов).

```
program Project1;
  {$APPTYPE CONSOLE}      {директива компилятора - создать консольное
                           приложение}

uses
  SysUtils, Windows;
var A, B, Summ : integer; {слагаемые и сумма}
    TmpStr : PChar;      {указатель на строку символов}
begin
  TmpStr := ''           // Инициализация TmpStr
                        {преобразование из кодировки 1251 (ANSI)
                        в кодировку 866 (OEM)}

  CharToOem('Введите значения двух целых чисел', TmpStr);
  Write(TmpStr);         {вывод сообщения в кодировке OEM}
  Readln(A, B);          {считывание с клавиатуры значений слагаемых}
  Summ := A + B;          {вычисление суммы}
```

```
WriteLn(Summ);           {вывод результата вычислений}
ReadLn;                  {останов программы до нажатия Enter
                           чтобы пользователь увидел результат}

end.
```

Сохраните проект в папке «Консольное приложение» под именем Summa. Откомпилируйте и запустите приложение на выполнение командой Run ► Run. Как показано на рис. 15.2, введите значения слагаемых и нажмите клавишу Enter.



Рис. 15.2. Окно MS DOS, в котором выполняется консольное приложение Summa

После просмотра результата вычислений нажмите клавишу Enter и окно MS DOS, в котором выполнялось консольное приложение, будет закрыто.

Попробуйте изменить текст программы, отключив использование функции CharToOem и организовав вывод сообщения оператором Write следующим образом:

```
{CharToOem('Введите значения двух целых чисел', TmpStr);}
Write('Введите значения двух целых чисел'); {вывод сообщения
                                             в кодировке ANSI}
```

Откомпилируйте и запустите приложение на выполнение, нажав клавишу F9. Как видно на рис. 15.3, на экране консольного приложения текст в кодировке Windows1251 (ANSI) будет напечатан непонятными символами.

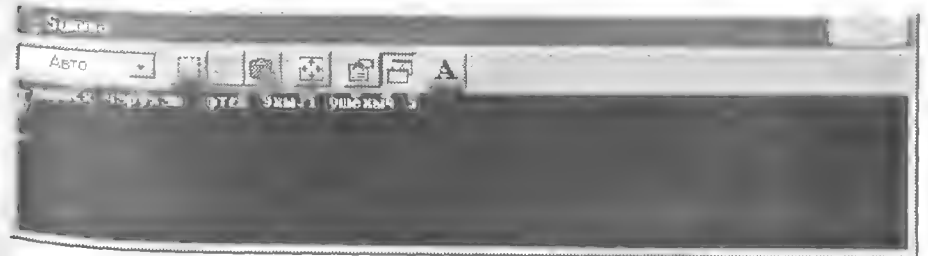


Рис. 15.3. Вид текста консольного приложения в кодировке Windows1251

Закройте окно MS DOS, в котором выполнялось консольное приложение, и отмените изменения, сделанные в программе, нажимая клавиши Ctrl+Z для отмены операции редактирования.

Многооконный текстовый редактор

Упражнение 2. Создайте многооконный текстовый редактор, который будет иметь следующие возможности: редактирование текста в нескольких окнах, изменение шрифта текста, выравнивание строк в абзаце, поиск и замена фрагментов текста, выделение, копирование и вставка фрагментов текста, сохранение текста и вывод на печать.

Запустите интегрированную среду разработки Delphi и создайте новое приложение командой **File ▶ New ▶ Application**. Переименуйте имя формы, выбрав в окне Инспектора объектов объект **Form1** и заменив значение свойства **Form1.Name** на **MainForm**. Задайте для свойства **MainForm.Caption** значение «Текстовый редактор». Чтобы окно редактора было родительским окном в MDI-приложении, задайте для свойства **MainForm.FormStyle** значение **fsMDIForm**.

ПРИМЕЧАНИЕ

MDI — это сокращение от Multiple Document Interface (интерфейс для одновременной работы с несколькими документами). В MDI-приложениях два или более окон могут быть активны одновременно. Наряду с MDI имеются приложения SDI (Single Document Interface, интерфейс для работы с одним документом). В SDI-приложениях в каждый момент времени может быть активным только одно окно. MDI-приложения являются удобным средством для одновременного вывода на экран текста или данных, которые хранятся в разных файлах. Такую структуру построения окон можно использовать для редактирования текстов, открывая и выводя на экран одновременно несколько разных документов. Пример такой работы с файлами представляет собой программа **Microsoft Word**.

Сохраните файлы проекта и модуля в папке «Многооконный текстовый редактор» под именами **Edit.dpr** и **main.pas**. Для создания меню приложения поместите на форму компонент **MainMenu** из палитры **Standard**. Дважды щелкните мышью по компоненту **MainMenu** на форме, чтобы раскрыть конструктор меню окна.

Далее, пользуясь свойством **Caption**, задайте для меню следующую структуру:

Файл	Правка	Формат	Окна	Помощь
Создать	Отменить	Шрифт	Каскад	О программе
Открыть	Вырезать	Выравнивание	Влево	Горизонтально
Сохранить	Копировать		По центру	Вертикально
Сохранить как	Вставить		Вправо	
Печать	Найти			
Выход	Заменить			

Чтобы во время работы приложения все дочерние окна отображались списком в меню **Окна**, выберите пункт меню **Окна** и переименуйте его в окне Инспектора объектов в **WindowMenu**. Затем, выбрав в окне Инспектора объектов форму **MainForm**, в свойстве **WindowMenu** выберите из раскрывающегося списка пункт меню **WindowMenu**.

СОВЕТ

Чтобы вызвать конструктор для размещенного в форме компонента главного меню (**MainMenu**), можно или дважды щелкнуть по нему мышью или дважды щелкнуть мышью по свойству **Items** в Инспекторе объектов.

Выбрать изменяемый пункт можно мышью, для изменения названия выберите в Инспекторе объектов свойство **Caption**, для отделения секций меню линией используйте знак тире в свойстве **Caption** для нужного пункта меню.

Для изменения, добавления или удаления пунктов в конструкторе меню, используя правую клавишу мыши, вызовите контекстное меню и выберите нужное действие.

Спроектируйте окно дочернего документа, для чего добавьте в проект новую форму, выбрав команду **File ▶ New ▶ Form**. Сделайте окно **Form2** меньше по сравнению с главным родительским окном и переименуйте свойство **Name** этого окна из **Form2** в **ChildForm**. Задайте для свойства дочернего окна **MDI-приложения ChildForm.FormStyle** значение **fsMDIChild**.

Поместите на форму **ChildForm** компонент **RichEdit** из палитры **Win32**. Чтобы компонент **RichEdit** занимал все окно **ChildForm**, и во время выполнения приложения его размеры изменялись при изменении размеров окна, измените свойство **RichEdit.Align** на **alClient**.

ПРИМЕЧАНИЕ

Компонент **TRichEdit** представляет собой многофункциональное средство редактирования текстов, позволяющее работать с форматом **.rtf**, т. е. выбирать различные атрибуты форматирования для разных фрагментов текста. В этом основное отличие **TRichEdit** от более простого компонента **TMemo**, в котором атрибуты форматирования одинаковы для всего текста. Окно редактирования снабжено многими функциями, собственными большинству редакторов. Например, в нем предусмотрены типичные комбинации клавиш оперативного доступа: **Ctrl+C** — копирование выделенного текста в буфер обмена (команда **Copy**), **Ctrl+X** — вырезание выделенного текста в буфер обмена (команда **Cut**), **Ctrl+V** — вставка текста из буфера обмена в позицию курсора (команда **Paste**), **Ctrl+Z** — отмена последней команды редактирования.

Дважды щелкнув мышью в поле значений свойства **RichEdit1.Lines**, в окне **String List Editor** удалите текст и нажмите **OK**.

Сохраните новый модуль под именем **Child.pas** в папке «Многооконный текстовый редактор» и сохраните изменения в проекте под прежним именем **Edit.dpr**. Нажатием **Shift+F12** выберите в окне **View Form** главную форму приложения **MainForm** и щелкните мышью на кнопке **OK**, как показано на рис. 15.4.

Поместите на главной форме **MainForm** компоненты из палитры **Dialogs**: **OpenDialog**, **SaveDialog**, **FontDialog**, **PrintDialog**, **FindDialog**, **ReplaceDialog** и **ActionList** из палитры **Standard**. Эти компоненты понадобятся в дальнейшем при обработке событий в нашем приложении. Так как они не являются визуальными компонентами, то их можно разместить в любом месте формы, как, например, показано на рис. 15.5.

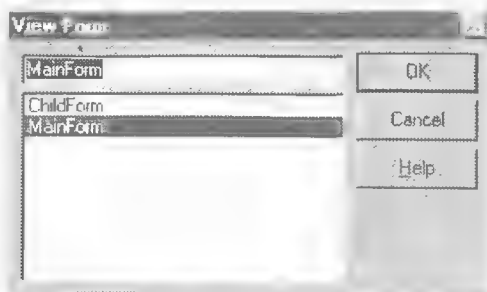


Рис. 15.4. Переход к форме MainForm

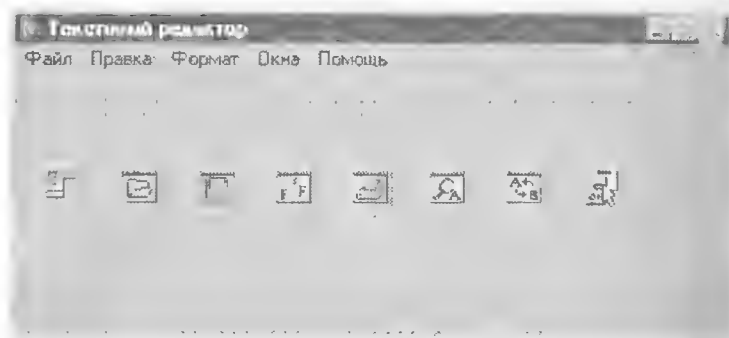


Рис. 15.5. Окно главной формы приложения с компонентами

Для создания дочернего окна документа в редакторе только при выборе меню **Файл** ► **Создать** отключите создание окна во время запуска приложения, которое задается Delphi по умолчанию. Для этого выберите в меню Delphi команду **Project** ► **Options...** В окне **Project Options** на вкладке **Forms** выберите форму **ChildForm** и, щелкнув мышью на кнопке со стрелкой, перенесите имя дочерней формы **ChildForm** из списка **Auto-create forms** (формы, автоматически создающиеся при старте приложения) в список **Available forms** (доступные формы), как показано на рис. 15.6. Для завершения операции щелкните мышью на кнопке **OK**.

Задайте для приложения значок с соответствующим изображением, например, блокнота с ручкой. Для этого присвойте соответствующее значение свойству **MainForm.Icon**, а также выберите этот значок при установке параметра **Application Icon** в окне **Project Options**.

Сохраните изменения в проекте командой **File** ► **Save All**.

Создайте процедуры обработчиков событий в приложении.

Для открытия нового дочернего окна приложения создайте процедуру обработчика выбора команды **Файл** ► **Создать**. Для этого выберите на форме **MainForm** в меню **Файл** команду **Создать**. При этом в программном модуле **Main** будет сгенерирована процедура обработчика события, которую нужно отредактировать следующим образом:

```
procedure TForm1.N2Click(Sender: TObject);
begin
    ChildForm.Create(Self) {создать дочернее окно документа}
end;
```

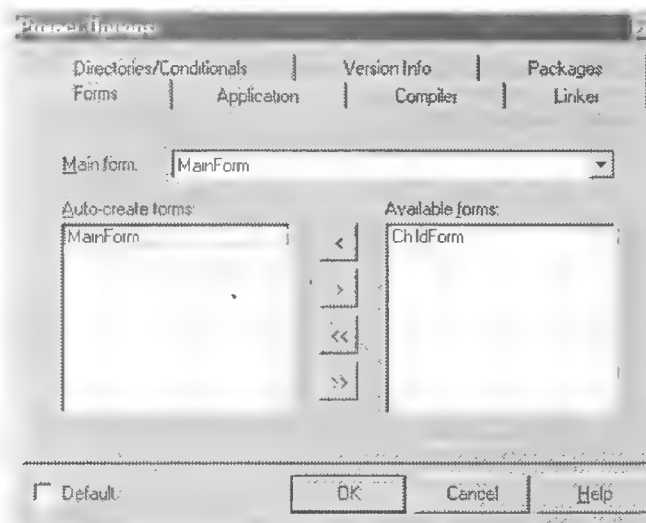


Рис. 15.6. Отключение создания окна ChildForm во время запуска приложения

Так как в этом окне создается другое окно, то к модулю **Main** необходимо подключить модуль **Child**. Для этого следует выбрать команду **File** ► **Use Unit...**, в окне **Use Unit** указать модуль **Child** и щелкнуть по кнопке **OK**.

Сохраните проект при помощи команды **File** ► **Save All**, откомпилируйте и запустите приложение на выполнение. Проверьте работу только что написанной процедуры обработчика события, выбирая в меню приложения команду **Файл** ► **Создать**. Убедитесь, что в окне приложения создается дочернее окно. Попробуйте закрыть дочернее окно **ChildForm**. Оно не закрывается, а минимизируется. Закройте окно приложения, щелкнув мышью на кнопке закрытия окна в правом верхнем углу окна.

Для того чтобы окно создаваемого документа имело название «Новый», создайте процедуру обработки события **OnCreate** для объекта **ChildForm**. Чтобы создать эту процедуру, откройте форму **ChildForm** и в окне Инспектора объектов на странице **Events** дважды щелкните мышью в поле значений события **OnCreate**, а затем добавьте оператор присваивания значения «Новый» свойству **Caption** создаваемого дочернего окна, как показано ниже:

```
procedure TForm1.ChildForm.Create(Sender: TObject);
begin
    Caption:='Новый'; // Заголовок окна нового файла
end;
```

Чтобы дочернее окно нормально закрывалось, перейдите в окно ChildForm (Shift+F12), затем выберите в окне Инспектора объектов объект ChildForm и на странице Events дважды щелкните мышью по пустому полю справа в строке события OnClose (закрытие окна). Отредактируйте заготовку процедуры закрытия окна следующим образом:

```
procedure TChildForm.FormClose(Sender: TObject; var Action: TCloseAction):
begin
    Action := caFree; {закрыть форму и уничтожить из памяти}
end;
```

ПРИМЕЧАНИЕ

Здесь вносимый в процедуру параметр Action определяет поведение данной формы при закрытии. Он может принимать следующие значения:

- caNone — форма не предпринимает каких-либо действий;
- caHide — форма делается скрытой, но не удаляется из памяти. Поскольку мы создаем дочернюю форму «вручную» (программно), то мы должны при ее закрытии удалить ее из памяти;
- caFree — форма закрывается и удаляется из памяти;
- caMinimize — форма минимизируется. При использовании MDI это значение для дочерних окон устанавливается по умолчанию.

Для создания процедуры закрытия окна редактора при выборе команды Файл ► Выход выберите эту команду в списке объектов Object TreeView и на странице Events в окне Инспектора объектов дважды щелкните мышью в поле значений события OnClick. После этого добавьте в процедуру обработки этого события оператор Close, так что текст процедуры будет выглядеть следующим образом:

```
procedure TMainForm.N8Click(Sender: TObject):
begin
    Close; {закрыть окно приложения}
end;
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Открывая и закрывая дочерние окна, убедитесь в правильности работы процедуры обработчика закрытия дочернего окна. Выбрав команду Файл ► Выход, завершите работу приложения.

Создавая процедуру открытия файла, учтите, что при выборе соответствующего пункта меню сначала открывается диалоговое окно выбора файла, и при успешном выборе создается форма ChildForm, в которой в компонент RichEdit1 загружается этот файл. Поэтому для организации открытия файла используйте компонент OpenFileDialog. Выделите объект в окне Инспектора объектов и задайте для него следующие свойства: для OpenFileDialog1.Title задайте значение «Открыть текстовый файл», а для свойства OpenFileDialog1.Filter задайте значения, как показано на рис. 15.7.

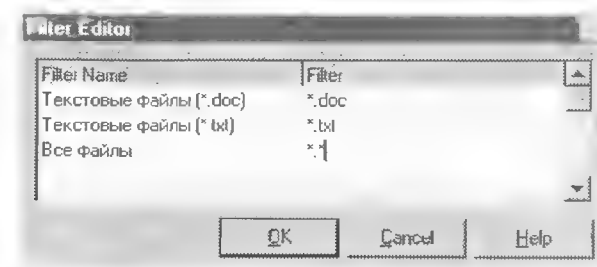


Рис. 15.7. Создание фильтра для OpenFileDialog1

Для создания процедуры открытия файла выберите в меню Файл на форме MainForm команду Открыть. После этого в окне редактора кода будет создана заготовка процедуры реакции на событие выбора этого меню. Отредактируйте ее следующим образом:

```
procedure TMainForm.N3Click(Sender: TObject):
begin
    if OpenFileDialog1.Execute then {если файл выбран,
                                    то выполнять следующее}
    begin
        ChildForm := TChildForm.Create(Self); {создать дочернее окно}
        ChildForm.RichEdit1.Lines.LoadFromFile(OpenFileDialog1.FileName);
        {загрузить в RichEdit1
        выбранный файл}
        ChildForm.Caption := OpenFileDialog1.FileName; {установить в заголовок
        дочернего окна название
        файла}
        ChildForm.RichEdit1.Tag:=0; {признак неизмененного
        файла}
    end;
end;
```

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Выбирая команду Файл ► Открыть, проверьте правильность выполнения процедуры открытия файла, открыв несколько файлов. Обратите внимание, что открытые окна располагаются в окне редактора каскадом. Выбрав в меню приложения пункт Окна, обратите внимание, что в меню окна отображаются имена дочерних окон документов, а при выборе имени документа в меню Окна, окно с этим документом становится активным и отображается на переднем плане.

Для того чтобы изменить порядок отображения файлов в окне приложения, необходимо создать процедуры обработки команд меню Окна.

Откройте главную форму приложения и дважды щелкните мышью по компоненту ActionList1 или выберите пункт Action List Editor из контекстного меню. В окне Editing MainForm1.ActionList1 нажмите Ctrl+Ins или, щелкнув мышью по

желтой кнопке в панели инструментов этого окна, выберите пункт New Standard Action из раскрывающегося меню для открытия окна Standard Action Classes со списком стандартных действий, которые можно привязать к пунктам меню или кнопкам. При нажатой клавише Shift или Ctrl выберите все стандартные действия, которые относятся к категории Window, как показано на рис. 15.8.

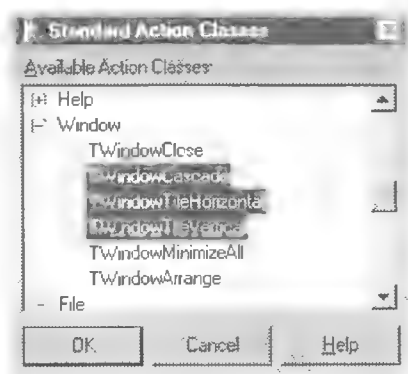


Рис. 15.8. Выбор списка стандартных действий управлением состоянием окон

Завершите выбор стандартных действий, щелкнув мышью на кнопке OK. После этого в списке действий появится категория Window, а в правой части окна будет показан список всех выбранных стандартных действий с окнами. Закройте окно Editing MainForm1.ActionList1 и убедитесь, что список выбранных стандартных действий отображается в окне Object TreeView, как показано на рис. 15.9.

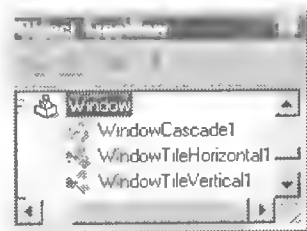


Рис. 15.9. Выбранные стандартные действия с окнами в окне Object TreeView

Укажите для каждой из команд меню Окна соответствующее стандартное действие. Выбрав в окне Object TreeView команду Каскад, в окне Инспектора объектов на странице Events откройте список значений стандартных событий Action и выберите в нем событие WindowCascade1, как показано на рис. 15.10.

Так как по умолчанию все стандартные действия имеют названия на английском языке, то после этой операции названия соответствующей команды меню поменяется на стандартное &Cascade. Чтобы заменить это название, на странице Properties выберите свойство Caption и замените его значение на &Каскад.

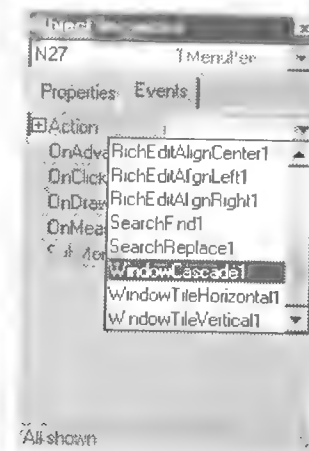


Рис. 15.10. Выбор стандартного события для команды меню

ПРИМЕЧАНИЕ

Знак амперсанд означает, что следующий за ним символ будет подчеркнут. Это свойство используется для быстрого выбора данного пункта меню с помощью оперативной клавиши, когда меню активизировано. Положение знака & указывает оперативную клавишу для выбора данной команды меню.

Аналогичным образом задайте стандартные действия для команд меню Окна Горизонтально и Вертикально, выбрав соответственно действия WindowTileHorizontal1 и WindowTileVertical1, а затем отредактировав значения их свойств Caption.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Выбором команды Файл ► Открыть откройте несколько файлов. Выбирая в меню Окна команды Каскад, Горизонтально или Вертикально, убедитесь в правильности размещения окон дочерних документов в окне приложения в соответствии с выбранной командой.

ПРИМЕЧАНИЯ

Обратите внимание, что если дочерних окон нет, то пункты меню Окна становятся недоступными.

Пример с подключением стандартных действий для обработки событий выбора команды в меню Окна наглядно иллюстрирует возможности создания приложения в визуальной объектно-ориентированной среде программирования Delphi. Вы не написали ни одной строки программного кода, а решили задачу, лишь изменяя некоторые свойства объектов в приложении.

Аналогичным образом подключите стандартные действия для обработки пунктов меню Правка (Отменить, Вырезать, Копировать, Вставить) и Формат ► Выравнивание (Влево, По центру, Вправо). При этом не забудьте о русификации свойства Caption.

Для создания процедуры изменения шрифта выберите на главной форме приложения команду **Формат** ► **Шрифт** и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TMainForm.N21Click(Sender: TObject): {изменение шрифта}
begin
  if FontDialog1.Execute then                {если пользователь выбрал
                                              шрифт}
    (ActiveMDIChild as TChildForm).RichEdit1.Font := FontDialog1.Font;
                                              {установить выбранный
                                              шрифт}
end;
```

Как видно из текста процедуры, сначала в диалоговом окне **Выбор шрифта** пользователь задает параметры шрифта, а затем, если эта операция выполнена успешно, выбранный шрифт присваивается свойству `RichEdit1.Font` окна документа.

ПРИМЕЧАНИЕ

Строка `(ActiveMDIChild as TChildForm)` позволяет получить доступ из главного окна к активному дочернему окну. Эта строка равноценна `ChildForm`, но выполняется она только для активного окна.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Создав новый документ, введите текст и проверьте действие команды **Формат** ► **Шрифт** для изменения шрифта в окне документа.

Чтобы документ, открытый в окне редактора, сохранялся в файле на диске, задайте для свойства `SaveDialog1.Filter` значение **(Текстовый файл (*.rtf) |*.rtf| Текстовый файл (*.txt) |*.txt)** и создайте процедуры обработчика события выбора в меню команды **Файл** ► **Сохранить как...** Для создания процедуры сохранения файла выберите на главной форме приложения команду **Файл** ► **Сохранить как...** и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TMainForm.N6Click(Sender: TObject): {сохранить как...}
begin
  if MDIChildCount=0 then Exit: {если нет MDI дочерних окон,
                                то выйти из процедуры}
  SaveDialog1.FileName := FName: {присвоить свойству FileName значение
                                  из FName}
  SaveDialog1.Title:='Сохранить файл как...': {заголовок диалогового
                                                окна сохранения файла}

  if SaveDialog1.Execute then
  begin
    FName:=SaveDialog1.FileName: {открыть диалог и запомнить
                                  новое имя файла}
    Case SaveDialog1.FilterIndex of {Изменить расширение файла}
```

```
1 . FName:=ChangeFileExt(FName,'.rtf');
2 . FName:=ChangeFileExt(FName,'.txt');
end:
end:
(ActiveMDIChild as TChildForm).RichEdit1.Lines.SaveToFile(FName):
  {записать в файл содержимое из свойства
  Lines объекта RichEdit активного окна}
(ActiveMDIChild as TChildForm).Caption:=FName:
  {заменить имя файла в заголовке активного окна}
(ActiveMDIChild as TChildForm).RichEdit1.Tag:=0;
  {изменения в файле сохранены}
end;
```

Как видно из текста процедуры, сначала проверяется, есть ли открытое окно документа. Если открытых окон документов нет, то процедура завершается. Если имеется открытое окно, то открывается диалоговое окно **Сохранить файл как...**, в котором задается вариант имени сохраняемого файла, взятый из переменной `FName`. В диалоговом окне пользователь может отредактировать имя и расширение файла. Если диалог завершается успешно, то запоминается присвоенное пользователем имя и расширение файла, а содержимое из свойства `Lines` объекта `RichEdit1` активного окна записывается в файл. После сохранения файла обновляется значение имени файла в заголовке окна документа, и свойству `Tag` объекта `RichEdit1` активного окна присваивается значение 0.

Для создания процедуры сохранения документа по команде **Файл** ► **Сохранить** выберите в окне на главной форме приложения команду **Файл** ► **Сохранить** и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TMainForm.N5Click(Sender: TObject): {сохранить файл}
begin
  if MDIChildCount=0 then Exit:
    {если нет MDI дочерних окон, то выйти из процедуры}
  if (ActiveMDIChild as TChildForm).Caption='Новый' then
    {если документ имеет заголовок окна «Новый»}
    N6Click(Sender) {вызвать процедуру «Сохранить как...»}
  else
  begin
    {Сохранить файл с именем, указанным в заголовке}
    (ActiveMDIChild as TChildForm).RichEdit1.Lines.
      SaveToFile((ActiveMDIChild as TChildForm).Caption):
    (ActiveMDIChild as TChildForm).RichEdit1.Tag:=0;
    {изменения в файле сохранены}
  end:
end;
```

Как видно из текста процедуры, она выполняет операцию сохранения файла, если окно документа не является пустым. Если документ имеет название

«Новый», то вызывается процедура обработчика события «Сохранить как...», текст которой был рассмотрен выше.

Чтобы проверять при закрытии окна, редактировался ли текст и если он изменялся, то предлагать сохранить файл, создайте процедуру обработки события закрытия окна документа. У каждого компонента в Delphi есть свойство Tag. Оно имеет тип Integer и является свойством, которым пользуются только программисты. Tag в программе ни на что не влияет и по умолчанию принимает нулевое значение. Как вы помните, при сохранении документа мы присваивали свойству Tag дочернего окна документа значение 0. Если документ в окне изменился, то нужно присвоить свойству Tag значение 1, чтобы указать на необходимость сохранения изменений при попытке закрыть окно.

Чтобы значение свойства Tag содержало информацию об изменениях в окне редактирования, создайте процедуру обработчика события OnChange для объекта RichEdit1 и включите в нее оператор присваивания RichEdit1.Tag:=1;

```
procedure TChildForm.RichEdit1Change(Sender: TObject);
begin
  RichEdit1.Tag:=1; {текст изменен и подлежит запросу на сохранение}
end;
```

Для создания процедуры закрытия окна документа с сохранением изменений откройте форму ChildForm и в окне Инспектора объектов на странице Events дважды щелкните мышью в строке события OnCloseQuery. Отредактируйте заготовку процедуры следующим образом:

```
procedure TChildForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  Var Res:Integer; {переменная для хранения результата выбора действия}
begin
  if RichEdit1.Tag=0 then CanClose:=True {если не сохранять файл, то}
  else {можно закрыть окно}
  begin
    {вопрос на сохранение}
    Res:=Application.MessageBox('Текст изменен.
    Сохранить изменения?'. 'Вопрос', MB_YESNOCANCEL + MB_ICONQUESTION);
    if Res=IDYES then
      begin {нажата кнопка «Да»}
        ChildForm.BringToFront. {расположить данное окно выше
        всех (сделать активным)}
        MainForm.Save(Sender); {вызвать процедуру «Сохранить»}
        if RichEdit1.Tag=1 then CanClose:=False;
        {если пользователь не сохранил
        файл, то окно не закрывать}
      end;
    if Res=IDNo then CanClose:=True;
    {если «Нет», то можно закрыть}
    if Res=IDCANCEL then CanClose:=False;
    {если «Отмена», то не закрывать}
  end;
end;
```

Как видно из текста процедуры, сначала проверяется значение свойства Tag. Если оно равно нулю, то документ в окне не изменялся и можно закрыть окно. Если Tag = 1, то в диалоговом окне отображается запрос на сохранение изменений и, в зависимости от выбора пользователя, определяемого значением параметра CanClose, выполнение операций: сохранение документа, выход без сохранения или отказ от закрытия окна.

Для того чтобы редактор обеспечивал вывод текстов на печать, необходимо создать процедуру обработчика события выбора команды Файл ► Печать. Для этого откройте форму MainForm и выберите эту команду, а затем в окне Редактора кода отредактируйте процедуру печати следующим образом:

```
procedure TMainForm.Print(Sender: TObject); {печатать текстовый файл}
begin
  if PrintDialog1.Execute then begin
    Assign(F); {объявить файловую переменную F - установить
    ассоциативную связь между файловой переменной
    F и принтером}
    Rewrite(F); {создать и открыть файл печати}
    Writeln(F.(ActiveMDIChild as TChildForm).RichEdit1.Text);
    {вывести текст из RichEdit1 на принтер}
    System.CloseFile(F); {закрыть файл печати}
  end;
end;
```

ВНИМАНИЕ

В раздел Uses необходимо добавить вызов модуля Printers.

Сохраните измененный проект, откомпилируйте и запустите приложение на выполнение. Открыв текстовый файл, проверьте печать текста командой Файл ► Печать.

Создайте процедуры поиска и замены фрагментов текста. Чтобы найденный фрагмент текста выделялся, задайте для свойства RichEdit1.HideSelection значение False. Для хранения позиции начала поиска введите в раздел описания переменных объявление целой переменной SPos.

Для вызова диалога поиска выберите на форме MainForm команду Правка ► Найти и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
{Процедура обработки события OnClick пункта меню «Найти»}
procedure TMainForm.N20Click(Sender: TObject); {вызов диалога поиска}
begin
  SPos:=(ActiveMDIChild as TChildForm).RichEdit1.SelStart;
  {запомнить позицию курсора}
  with FindDialog1 do
  begin
    FindText:=(ActiveMDIChild as TChildForm).RichEdit1.SelText;
```

```

        {начальное значение искомого текста - выделенный текст}
        {позиционирование окна диалога поиска}
        Position:=Point(ChildForm.Left,ChildForm.Top+
            ChildForm.RichEdit1.Top+ChildForm.RichEdit1.Height);
        Options:=Options+[frHideUpDown,frHideWholeWord];
        {удаление из диалога кнопок «Вверх», «Вниз»,
        «только слово целиком»}
        Execute:      {выполнение поиска}
    end:
end:

```

Для того чтобы при щелчке мышью на кнопке Найти далее в окне Поиск выполнялся поиск заданного фрагмента текста, создайте процедуру обработки события OnFind объекта FindDialog1 и отредактируйте ее следующим образом:

```

{Процедура обработки события OnFind объекта FindDialog1}
procedure TMainForm.FindDialog1Find(Sender: TObject);
begin
    with FindDialog1 do
        begin
            if frMatchCase in Options then    {поиск с учетом регистра}
                ChildForm.RichEdit1.SelStart:=Pos(FindText,
                    Copy(ChildForm.RichEdit1.Lines.Text,SPos+1,
                        Length(ChildForm.RichEdit1.Lines.Text)))+SPos-1
            else                               {поиск без учета регистра}
                ChildForm.RichEdit1.SelStart:=Pos(AnsiLowerCase(FindText),
                    AnsiLowerCase(Copy(ChildForm.RichEdit1.Lines.Text,SPos+1,
                        Length(ChildForm.RichEdit1.Lines.Text)))+SPos-1;
            if ChildForm.RichEdit1.SelStart>=SPos then
                begin                               {выделение найденного текста}
                    ChildForm.RichEdit1.SelLength:=Length(FindText);
                    SPos:=ChildForm.RichEdit1.SelStart+
                        ChildForm.RichEdit1.SelLength+1;
                end
            else
                if MessageDlg('Текст «'+FindText+'» не найден.
                    Продолжать диалог?',mtConfirmation,mbYesNoCancel,0)<@060>=mrYes
                    then CloseDialog;
            end;
            ChildForm.RichEdit1.SetFocus;
        end;
    end;
end;

```

Для вызова диалога замены выберите на форме MainForm команду Правка ► Заменить и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

{процедура вызова диалога замены: обработка события OnClick пункта меню
    Заменить»}
procedure TMainForm.N19Click(Sender: TObject);

```

```

begin
    SPos:=ChildForm.RichEdit1.SelStart;      {запомнить позицию курсора}
    with ReplaceDialog1 do
        begin
            FindText:=ChildForm.RichEdit1.SelText; {начальное значение
                                                    искомого текста -
                                                    выделенный текст}
                                                    {позиционирование окна
                                                    диалога поиска}

            Position:=Point(ChildForm.Left,ChildForm.Top+
                ChildForm.RichEdit1.Top+ChildForm.RichEdit1.Height);
            Options:=Options+[frHideUpDown,frHideWholeWord,frReplaceAll];
                                                    {удаление из диалога кнопок}
            Execute:      {выполнение поиска-замены}
        end;
    end;
end;

```

Чтобы при щелчке на кнопке Заменить в окне Замена выполнялся поиск фрагмента текста по образцу и замена его на заданный, создайте процедуру обработки события OnReplace объекта ReplaceDialog1 и отредактируйте ее следующим образом:

```

{Процедура обработки события OnReplace объекта ReplaceDialog1}
procedure TMainForm.ReplaceDialog1Replace(Sender: TObject);
begin
    with TReplaceDialog(Sender) do
        begin
            SPos := Pos(FindText, ChildForm.RichEdit1.Lines.Text);
            if SPos > 0 then
                begin                               {если есть искомый текст}
                    ChildForm.RichEdit1.SelStart := SPos - 1;
                    ChildForm.RichEdit1.SelLength := Length(FindText);
                    ChildForm.RichEdit1.SelText := ReplaceText; {замена текста}
                end
            else MessageDlg('Текст "'+FindText+'" не найден.',mtError,
                [mbOk],0);
        end;
    end;
end;

```

Сохраните текст программы, откомпилируйте его и проверьте работу процедур поиска и замены.

Чтобы при выборе в меню приложения команды Помощь ► О программе вывести сведения о программе, создайте новое окно модального типа, открыв которое пользователь не сможет работать с другими окнами, пока не закроет окно сообщения О программе. Создайте новое окно командой File ► New ► Form. Установите размер новой формы и измените значение свойства Name на About. Задайте для свойства About.Caption значение «О программе». Выбрав команду File ► Save All, сохраните новое окно под именем AboutUnit. Чтобы окно сообще-

ния данных о программе открывалось посередине экрана, задайте для свойства `About.Position` значение `poScreenCenter`. Чтобы пользователь не мог свернуть это окно или развернуть его на весь экран, задайте свойству `About.BorderStyle` значение `bsDialog`.

Поместите на форму `About` кнопку `TbitBtn` из палитры компонентов `Additional`. Чтобы окно `About` закрывалось при выборе пользователем этой кнопки без написания каких-либо процедур с вашей стороны, задайте для свойства `About.Kind` значение `bkOK`. После этого кнопка приобретет новое название `OK` и на ней появится рисунок зеленой галочки.

Разместите на форме `About` компоненты `Label1` и `Label2`. Задайте для их свойств `Caption` значения «Пример текстового редактора» и «© 2002 Версия 1.0». Задайте шрифт в свойствах `Label1`.

Выровняйте и зафиксируйте компоненты на форме, как показано на рис. 15.11.

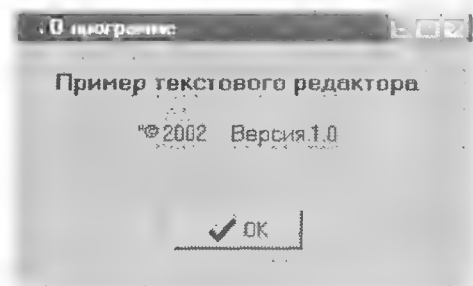


Рис. 15.11. Вид формы `About`

Создайте процедуру открытия окна `About` при выборе команды `Помощь ► О программе`. Для этого перейдите на форму `MainForm` и выберите команду `Помощь ► О программе`, а затем в заготовке процедуры введите оператор вызова окна сообщения модального типа `About.ShowModal`; так что текст процедуры будет выглядеть следующим образом:

```
procedure TMainForm.N31Click(Sender: TObject);
begin
    About.ShowModal;
end;
```

Чтобы подключить к проекту модуль `AboutUnit`, выберите команду `File ► Use Unit` и в окне `Use Unit` выберите модуль `AboutUnit`.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Проверьте появление окна сообщения `О программе` при выборе команды `Помощь ► О программе`.

Итак, вы создали многооконный текстовый редактор, который, конечно, можно будет в дальнейшем усовершенствовать.

Взаимодействие приложения с внешними программами

Создавая приложения в ИСП Delphi, вы имеете возможность организовать следующие взаимодействия вашего приложения с внешними программами:

- непосредственный запуск внешней программы из приложения;
- запуск внешней программы, связанной с некоторым документом;
- обмен сообщениями с другими программами;
- внедрение и связывание документов, подготовленных внешними программами, с вашим приложением;
- динамический обмен данными между приложениями.

Рассмотрим пример, в котором из вашего приложения будут запускаться различные внешние программы. Для запуска внешних программ можно использовать функцию `WinExec` или `ShellExecute`.

Функция `WinExec` позволяет выполнить указанное приложение. Определение функции: `function WinExec(CmdLine: PChar; CmdShow: integer): integer;`

Параметр `CmdLine` является указателем на строку с нулевым символом в конце, содержащую имя исполняемого файла и, если необходимо, параметры командной строки. Если имя указано без пути, то Windows ищет выполняемый файл в следующей последовательности.

1. В каталоге, из которого запущено приложение.
2. В текущем каталоге.
3. В системном каталоге Windows, имя которого возвращается функцией `GetSystemDirectory`.
4. В каталоге Windows, имя которого возвращается функцией `GetWindowsDirectory`.
5. В каталогах из списка в переменной окружения `PATH`.

Параметр `CmdShow` определяет форму представления окна запускаемого приложения Windows. Возможные значения этого параметра см. в разделе `ShellExecute`. Для приложений, не являющихся Windows-приложениями, файлов `PIF` и т. д., состояние окна определяет само приложение.

При успешном выполнении запуска приложения функция `WinExec` возвращает значение, большее 31.

Достоинством функции `WinExec` является ее совместимость с ранними версиями Windows. Собственно, для этого она и сохраняется в WIN32, хотя для Win32 рекомендуется пользоваться функцией `CreateProcess`. При работе с Win32 функция `WinExec` завершает работу, если вызванное приложение вызывает функцию `GetMessage` или заканчивается выделенный лимит времени. Таким образом, ожидание можно прервать, предусмотрев в процессе, запущенном с помощью `WinExec`, в нужный момент вызов функции `GetMessage`.

В отличие от WinExec, функция ShellExecute позволяет не только выполнить любое приложение Windows, но и открыть файл документа, что означает выполнение связанного с ним приложения и загрузку в него этого документа. Например, обычно с документами, имеющими расширение .doc, связана программа WinWord. В этом случае открыть файл, например, с именем file.doc означает запустить WinWord и передать ему в качестве параметра имя файла file.doc. Кроме описанных возможностей функция ShellExecute позволяет распечатать указанный файл или открыть указанную папку. Последнее означает, что будет запущена программа «Проводник» с открытой указанной папкой. Определение функции:

```
function ShellExecute(Wnd: HWnd; Operation: PChar; FileName: PChar;
  Parameters: PChar; Directory: PChar; ShowCmd: Integer): THandle;
```

Важнейшие параметры функции имеют следующие значения:

Параметр	Описание
Wnd	Родительское окно, в котором отображаются сообщения запускаемого приложения
Operation	Указывает на строку с нулевым символом в конце, которая определяет выполняемую операцию. Эта строка может содержать текст «open» (открыть) или «print» (напечатать). Для Windows 95 и NT определено еще одно значение: «explore» (открыть папку). Если параметр Operation равен nil, то по умолчанию выполняется операция «open»
FileName	Указывает на строку с нулевым символом в конце, которая определяет имя открываемого файла
Parameters	Указывает на строку с нулевым символом в конце, которая определяет передаваемые в приложение параметры, если FileName определяет выполняемый файл. Если FileName указывает на строку, определяющую открываемый документ, то этот параметр равен nil
Directory	Указывает на строку с нулевым символом в конце, которая определяет каталог, используемый по умолчанию
ShowCmd	Определяет режим открытия указанного файла

Функция ShellExecute возвращает дескриптор открытого приложения или дескриптор сервера DDE-приложения. Если возвращаемое значение меньше или равно 32, это указывает на ошибку.

Упражнение 3. Создайте приложение, из которого, используя команды меню, можно вызвать внешние программы Калькулятор, текстовый процессор Microsoft Word или табличный процессор Microsoft Excel.

Создайте новый проект и задайте для свойства Form1.Caption значение «Пример запуска внешних программ». Поместите на форму компонент MainMenu из палитры компонентов Standard. Дважды щелкните мышью по компоненту MainMenu на форме, чтобы раскрыть конструктор меню окна. Далее, пользуясь свойством Caption, задайте меню следующего вида, как показано на рис. 15.12.

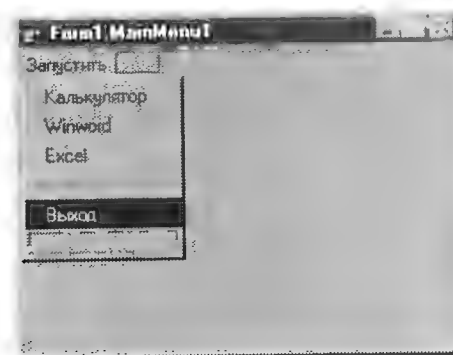


Рис. 15.12. Вид меню приложения в окне конструктора меню

Закройте окно конструктора-меню и, открыв окно Редактора кода, добавьте в раздел uses вызов модуля ShellAPI, в котором содержится функция запуска внешней программы ShellExecute, так как Delphi автоматически не включает этот модуль.

Откройте форму приложения и, выбрав команду меню Запустить ► Выход, создайте процедуру завершения работы приложения, записав в теле процедуры оператор Close; следующим образом:

```
procedure TForm1.N4Click(Sender: TObject);
begin
  Close; {закрыть окно приложения}
end;
```

Для создания процедуры обработки выбора команды меню Запустить ► Калькулятор выберите эту команду в меню, а затем отредактируйте текст процедуры-обработчика следующим образом:

```
procedure TForm1.N2Click(Sender: TObject); {вызов калькулятора}
begin
  ShellExecute(Handle, 'open', 'Calc', nil, nil, SW_Restore);
end;
```

Как видно из оператора вызова функции ShellExecute, параметр «open» указывает на операцию «Открыть» приложение с именем файла «Calc». Значение Parameters=nil указывает, что параметр FileName определяет исполняемый файл. Значение Directory=nil указывает на строку с нулевым символом в конце, которая определяет каталог, используемый по умолчанию. Параметр SW_RESTORE команды ShowCmd активизирует и отображает окно запускаемого приложения. Если это окно свернуто или развернуто, то оно восстанавливается до своих первоначальных размеров и отображается в первоначальной позиции.

Сохраните проект под именем ExecProgram, откомпилируйте приложение и запустите его на выполнение. Выбрав в меню вашего приложения команду Запустить ► Калькулятор, проверьте запуск внешней программы (рис. 12.13).

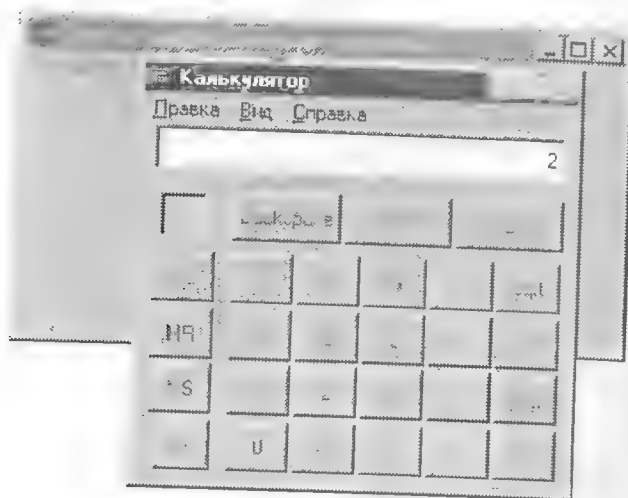


Рис. 15.13. Выполнение внешней программы, вызванной из приложения

Закройте окно приложения Калькулятор и завершите работу приложения, выбрав команду Запустить ► Выход.

Откройте форму приложения и, выбрав команду меню Запустить ► Winword, создайте процедуру запуска из вашего приложения текстового процессора Microsoft Word, отредактировав заготовку процедуры следующим образом:

```
procedure TForm1.Winword1Click(Sender: TObject); {вызов Microsoft Word}
begin
    WinExec('C:\Program Files\Microsoft Office\Office\Winword.exe'
    SW_RESTORE);
end;
```

Для создания процедуры запуска из вашего приложения табличного процессора Microsoft Excel откройте форму приложения и, выбрав команду меню Запустить ► Excel, создайте заготовку процедуры, а затем вставьте в нее вызов функции WinExec, указав в качестве файла внешней программы файл Excel.exe с параметром SW_RESTORE (активизировать и отобразить окно запускаемого приложения) следующим образом:

```
procedure TForm1.Excel1Click(Sender: TObject); {вызов Microsoft Excel}
begin
    WinExec('C:\Program Files\Microsoft Office\Office\Excel.exe',
    SW_RESTORE);
end;
```

Сохраните изменения в проекте, откомпилируйте приложение и запустите его. Выбирая команды Запустить ► Калькулятор, Запустить ► Winword, Запустить ► Excel, проверьте правильность вызова внешних программ, затем закройте окна внешних программ и созданного вами приложения.

Приложение для работы с базами данных

Как было рассказано в главе 12, интегрированная среда разработки Delphi обеспечивает возможность создания приложений для работы с локальными и удаленными базами данных любых типов.

Упражнение 4. Создайте приложение, которое выполняет обслуживание реляционной базы данных «Памятники Московского кремля», обеспечивая просмотр, добавление и удаление записей.

ПРИМЕЧАНИЕ

Предполагается, что читатель знаком с основными понятиями реляционных баз данных.

Создание новой таблицы в Database Desktop

Для хранения информации о памятниках Московского кремля создайте базу данных Paradox7 с таблицей следующей структуры:

Имя поля	Тип	Размер	Назначение
Name	Символьный	30	Название памятника
Year	Символьный	15	Год создания
Author	Символьный	30	Авторы
Comment	Примечаний (Мемо)		Краткое описание
Foto	Графический (Graphic)		Фотография памятника

Так как база данных в Paradox 7 — это папка, в которой расположены таблицы, то сначала создайте папку проекта, например, Kreml, а затем в этой папке создайте папку, например, Data, для размещения в ней таблиц базы данных.

Чтобы запустить поставляемую в комплекте с Delphi программу Database Desktop, выберите в меню Delphi команду Tools ► Database Desktop или выберите в главном меню Windows команду Пуск ► Программы ► Borland Delphi ► Database Desktop.

Для создания таблицы выберите в меню Database Desktop команду File ► New ► Table, а затем в окне Create Table (Создание таблицы) выберите тип таблицы Paradox 7 и щелкните мышью на кнопке OK.

В окне Create Paradox 7 Table опишите структуру данных. Щелкнув мышью в поле Field Name (Имя поля), введите имя поля, например Name. Завершите ввод имени поля, нажав клавишу Enter. После этого фокус ввода переместится в столбец Type (Тип поля). Щелкнув клавишу Space (Пробел) или, щелкнув правой кнопкой мыши, откройте список типов данных и выберите из него тип данных Alpha (Строка символов) и нажмите Enter. Нажатием клавиши Tab перейдите в столбец Size (размер) и задайте максимальную длину строки символов, например, 30. Нажав клавишу со стрелкой вниз, перейдите к описанию параметров следующего поля. Аналогичным образом опишите параметры полей Year, Author, Comment, Foto, как показано на рис. 15.14.

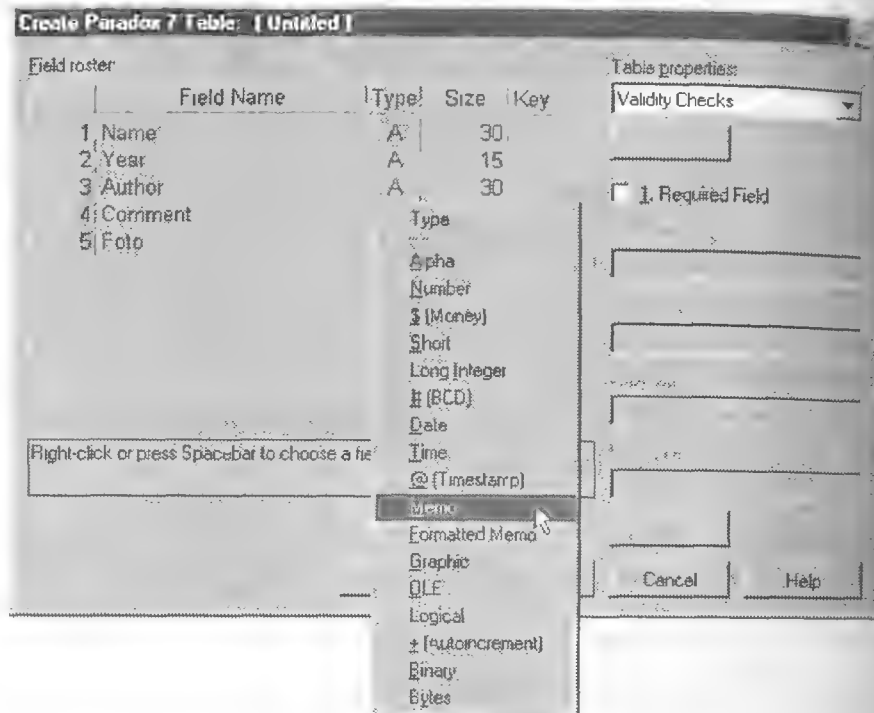


Рис. 15.14. Определение типа поля

ПРИМЕЧАНИЯ

Для поля Comment задайте тип Мемо — поле примечаний переменной длины. Чтобы иметь возможность хранить в поле Foto изображения, выберите для него тип Graphic. В правой части окна можно задать дополнительные параметры, такие как проверка правильности значений данных, таблица просмотра, целостность ссылок, вторичный индекс, пароль доступа, язык таблицы и др.

После описания структуры таблицы щелкните мышью на кнопке Save As и в окне Save Table As выберите в поле Сохранить в: папку для размещения таблиц базы данных, в нашем примере, Data. Затем в поле Имя файла: задайте имя файла таблицы, например, Kreml. В раскрывающемся списке Alias оставьте неизменным псевдоним баз данных None, как показано на рис. 15.15. В дальнейшем мы рассмотрим изменение псевдонима базы данных.

ПРИМЕЧАНИЕ

Псевдоним (alias) содержит всю информацию, необходимую для обеспечения доступа к базе данных.

ПРИМЕЧАНИЕ

Для других СУБД создание базы данных и диалоги сохранения отличаются с учетом особенностей различных СУБД.

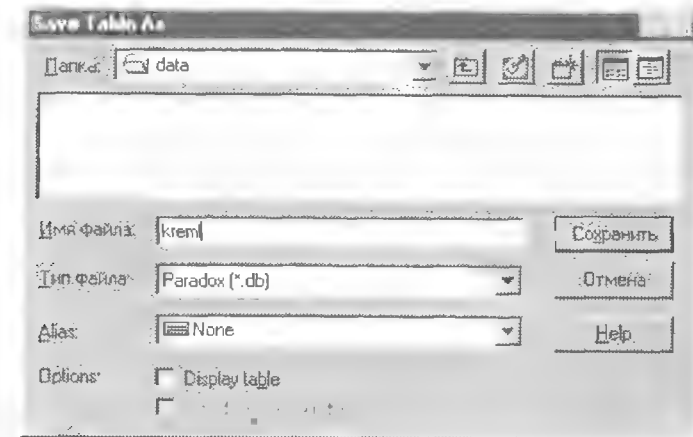


Рис. 15.15. Сохранение таблицы базы данных

Откройте созданную таблицу в окне Database Desktop командой File ► Open ► Table. Вид созданной таблицы показан на рис. 15.16.

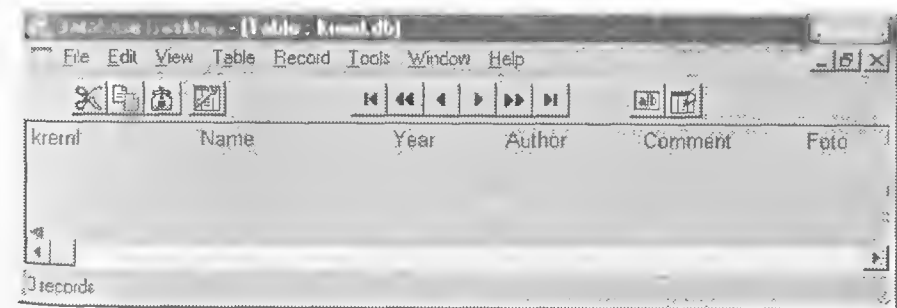


Рис. 15.16. Вид созданной таблицы в окне Database Desktop

Для ввода и редактирования данных в таблице можно выбрать в меню Database Desktop команду Table ► Edit или нажать клавишу F9. Установив курсор в нужную ячейку таблицы, введите значение поля. Нажатием клавиши Tab перейдите к вводу данных в следующее поле записи. Для добавления записи в таблицу выберите команду Record ► Insert. Остальные данные введите позднее в SQL Explorer.

Создание псевдонима базы данных

Прежде чем приступить к заполнению таблицы в SQL Explorer, создайте псевдоним базы данных, для чего запустите SQL Explorer, выбрав в меню Delphi команду Database ► Explore. Для создания псевдонима (Alias) в меню SQL Explorer выберите команду Object ► New, затем в окне New Database Alias выберите тип драйвера STANDARD и щелкните мышью на кнопке OK. По умолчанию создаваемому псевдониму будет присвоено имя STANDARD1. Затем

в правой части окна щелкните в строке PATH для выбора каталога с таблицами базы данных Paradox. В окне Select Directory выберите папку с файлами таблиц базы данных, как показано на рис. 15.17, и щелкните мышью по кнопке OK.

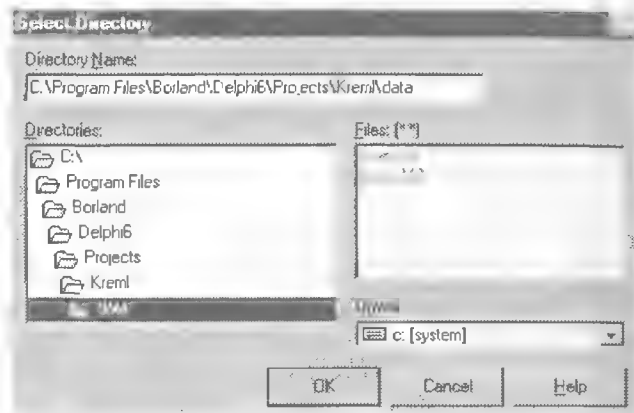


Рис. 15.17. Выбор каталога базы данных

Для сохранения созданного псевдонима базы данных выберите в меню SQL Explorer команду Object ► Save As, а затем в окне Save STANDARD1 as задайте имя псевдонима, например, Monument, и щелкните мышью на кнопке OK.

Заполнение новой таблицы в SQL Explorer

Чтобы открыть таблицу базы данных с созданным псевдонимом в SQL Explorer, выберите на вкладке Databases в списке псевдонимов псевдоним Monument и, щелкнув мышью по символу «+», разверните список таблиц базы данных. Выбрав таблицу kreml.db, в правой части окна на вкладке Data введите данные, как показано на рис. 15.18.

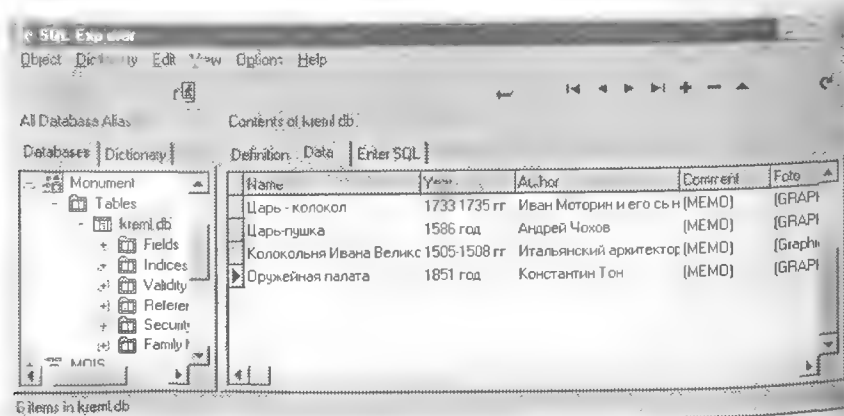


Рис. 15.18. Выбор базы данных и ввод данных в SQL Explorer

ПРИМЕЧАНИЕ

Для ввода данных в поля типа Memo и Graphics дважды щелкните левой кнопкой мыши, а затем введите в раскрытое окно текст или вставьте изображение из буфера обмена.

Завершите работу в SQL Explorer, выбрав в его меню команду Object ► Exit и подтвердите сохранение данных, щелкнув мышью на кнопке Yes в окне сообщения Save all edits to SQL Explorer.

Создание формы приложения

Создайте форму и разместите на ней компоненты: DataSource из палитры Data Access, Table из палитры BDE, DBNavigator из палитры DataControls.

ПРИМЕЧАНИЯ

Компонент TDataSource представляет собой источник данных, который обеспечивает связь между набором данных и компонентами отображения и редактирования данных. TTable — компонент набора данных, связанный с одной таблицей. Он обеспечивает прямой доступ к каждой записи и полю в одной указанной таблице базы данных. TDBNavigator — компонент, предназначенный для перемещения по набору данных и выполнения редактирования данных. TDBNavigator имеет ряд кнопок, служащих для управления данными. Пользуясь свойством навигатора VisibleButtons, можно исключить любые ненужные в данном приложении кнопки.

ВНИМАНИЕ

Чтобы приложение с навигатором работало, надо установить его основное свойство DataSource — источник данных (имя компонента TDataSource).

Задайте для свойства Form1.Caption значение «Памятники московского Кремля». Для связывания компонентов DataSource1, Table1 и DBNavigator1 в окне Инспектора объектов задайте следующие значения свойств:

DataSource1.Name = DataSource1

DataSource1.DataSet = Table1

Table1.DatabaseName = Monument (псевдоним БД)

Table1.TableName = kreml.db (имя базы данных)

Table1.Active = True (включить отображение данных)

DBNavigator1.Align = alBottom (отображать в нижней строке окна)

DBNavigator1.DataSource = DataSource1

Для отображения полей записи базы данных в определенном месте формы, дважды щелкнув мышью по компоненту Table1, откройте окно Form1.Table1 со списком полей таблицы. Для отображения полей, щелкнув правой кнопкой мыши, в контекстном меню выберите Add fields (Добавить поля), как показано на рис. 15.19.

Выбрав в списке Add fields поля, которые требуется отображать на форме, нажмите OK. Переместите каждое поле из списка в окне Form1.Table1 в опре-

деленное место на форме. Если для Table1.Active задано значение True, то в поле немедленно отображается значение соответствующего атрибута первой записи базы данных. Разместите и зафиксируйте положение визуальных компонентов с отображением полей записи базы данных, например, как показано на рис. 15.20.

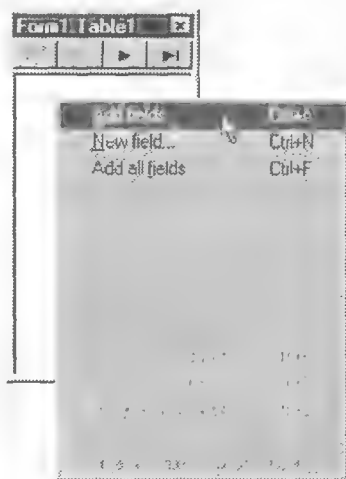


Рис. 15.19. Добавление полей записи базы данных в список отображаемых на форме

Чтобы в окне с краткой информацией о памятнике можно было прочитать текст большой длины, используя вертикальную полосу прокрутки, для объекта DBMemo1 задайте свойство ScrollBars=ssVertical.

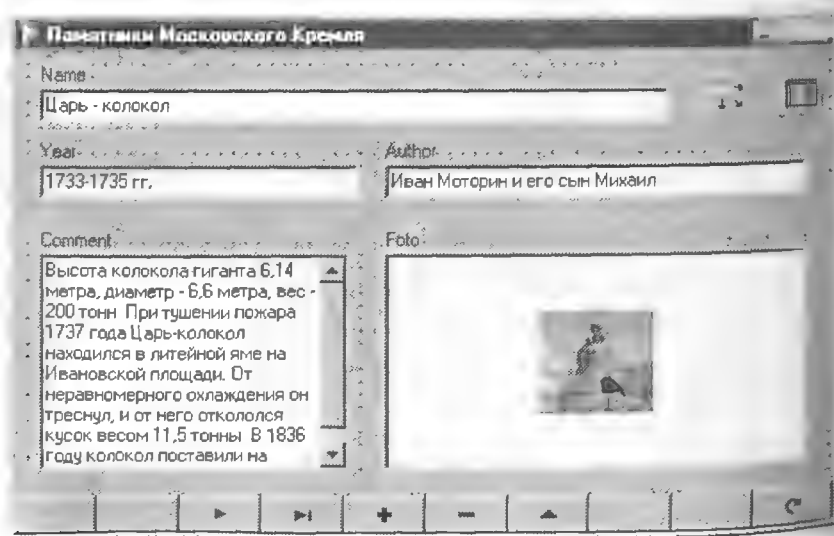


Рис. 15.20. Вид формы приложения для работы с базой данных

Используя свойство Caption объектов Label 1-5, измените тексты подписей к полям базы данных, вместо «Name» введите «Название», вместо «Year» — «Годы создания», вместо «Author» — «Авторы», вместо «Comment» — «Краткая справка», вместо «Foto» — «Фотография».

Сохраните проект, откомпилируйте его и запустите приложение. Щелкая мышью по кнопкам DBNavigator, как показано на рис. 15.21, проверьте переход к первой, предыдущей, следующей, последней записи. Щелкнув мышью на кнопке nbInsert со знаком +, вставьте новую запись перед текущей и введите данные, например, об Успенском соборе: «Успенский собор был построен в 1479 г. На протяжении шести столетий собор являлся государственным и культовым центром России: здесь ставили великих князей, а удельные присягали им на верность, венчали на царство, короновали императоров. В Успенском соборе возводили в сан епископов, митрополитов и патриархов, оглашали государственные акты». Фотографию собора вы можете найти в Интернете.

Используя Image Editor, создайте файл значка приложения; для этого в меню Image Editor выберите команду File ► New ► Icon File(*.ico). В окне Icon Properties (Свойства значка) выберите Size: 32 x 32, Colors: 16 color. Нарисуйте в редакторе значок, например, изображение Царь-колокола. Сохраните созданный значок под именем kreml.ico в папке проекта. Закройте окно Image Editor.

Выбрав в окне Инспектора объектов объект Form1, щелчком в поле значений свойства Icon откройте окно Picture Editor. Щелкните мышью на кнопке Load, в окне Load Picture выберите требуемый файл значка и щелкните на кнопке Открыть. Просмотрев изображение значка, нажмите кнопку OK. Вид значка на форме изменится. Чтобы изменить значок приложения, выберите в меню Delphi команду Project ► Options. На вкладке Application, щелкнув на кнопке Load Icon, загрузите созданный значок.

Откройте окно Редактора кода и убедитесь, что в процессе проектирования формы в окне Редактора кода был сгенерирован код программы.

Чтобы соединение и разрыв соединения с базой данных выполнялись с началом и окончанием работы приложения, задайте для свойства Active объекта Table1 значение False и создайте процедуры обработки событий OnCreate и OnDestroy для объекта Form1. В тело процедуры обработчика события создания формы добавьте оператор Table1.Active:= True; , а в тело обработчика события уничтожения формы добавьте оператор Table1.Active:= False; .

После этого текст программного модуля будет выглядеть следующим образом:

```
unit Main
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, ExtCtrls, DBCtrls, StdCtrls, Mask;
type
```

```

TForm1 = class(TForm)
  DBNavigator1: TDBNavigator;
  Table1: TTable;
  DataSource1: TDataSource;
  Label1: TLabel;
  DBEdit1: TDBEdit;
  Label2: TLabel;
  DBEdit2: TDBEdit;
  Label3: TLabel;
  DBEdit3: TDBEdit;
  Label4: TLabel;
  DBMemo1: TDBMemo;
  Label5: TLabel;
  DBImage1: TDBImage;
  TableName: TStringField;
  TableYear: TStringField;
  TableAuthor: TStringField;
  TableComment: TMemoField;
  TableFoto: TGraphicField;
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject); {создание формы}
begin
  Table1.Active:= True;                      {соединение с базой данных}
end;
procedure TForm1.FormDestroy(Sender: TObject); {уничтожение формы}
begin
  Table1.Active:= False;                     {разрыв соединения с базой данных}
end;
end.

```

Сохраните изменения в проекте, откомпилируйте и запустите приложение. Используя кнопки DBNavigator, как показано на рис. 15.21, проверьте работу приложения.

Конечно, созданное в данном упражнении простейшее приложение иллюстрирует лишь малую часть возможностей Delphi по разработке приложений для работы с базами данных.

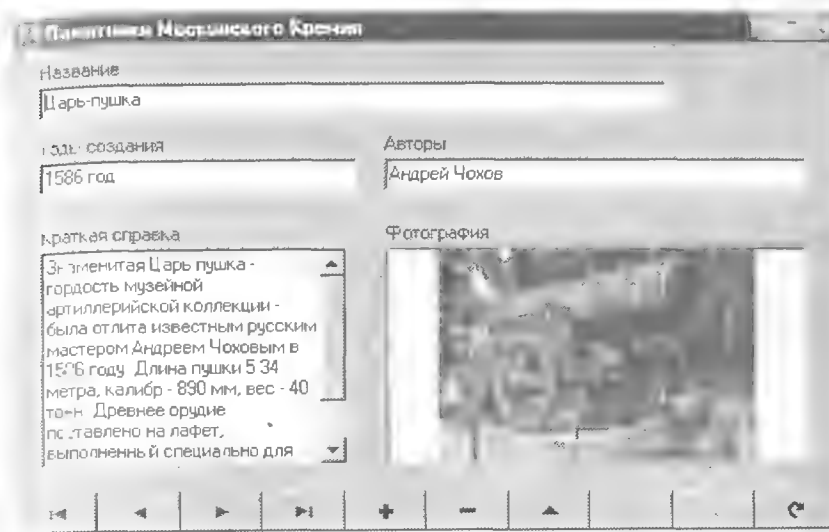


Рис. 15.21. Вид окна приложения для работы с базой данных

Итак, уважаемый читатель, мы с вами сделали первые шаги в освоении одной из современных сред разработки приложений. Успехов вам в дальнейшем совершенствовании знаний возможностей Delphi и практических навыков разработки приложений!

Приложение А

Глоссарий

А

Адрес — идентификатор места в памяти ЭВМ, где хранится или куда может быть записана информация.

Активный диалог — режим взаимодействия пользователя и программной системы, который характеризуется равноправием его участников. Обычно для организации активного диалога используются директивные (командные) языки, или языки, близкие к естественным.

Активное окно — окно, с которым в настоящий момент работает или может работать пользователь.

Актуализация данных — приведение данных в соответствие с состоянием отображаемых объектов предметной области. Актуализация реализуется посредством операций добавления, исключения и редактирования записей.

Алгоритм (Algorithm) — четко определенная последовательность действий для достижения поставленной цели за конечное число шагов.

Алфавит — фиксированный для каждого алгоритмического языка набор основных символов, из которых должен состоять любой текст на этом языке. Использование никаких других символов в тексте не допускается.

Американский стандартный код обмена информацией (ASCII) — стандартная схема кодирования текстовой информации, при которой каждый текстовый или управляющий символ представляется в виде семипразрядного двоичного кода.

Анимация — создание впечатления движения объекта на экране дисплея. В основе анимации лежит быстрая смена последовательно смещаемых друг относительно друга образов.

Аппаратные средства (Hardware) — механические, электрические, электронные узлы и компоненты компьютера.

Аппаратные средства мультимедиа — основные средства: компьютер с высокопроизводительным процессором и памятью большого объема, манипуляторами и мультимедиа-монитором со встроенными стереодинамиками; специальные средства: приводы CD-ROM, TV-тюнеры и карты видеозахва-

та, графические ускорители, платы видеовоспроизведения, звуковые платы, акустические системы и др.

Аппаратное прерывание — прерывание, причиной которого является событие, произошедшее в аппаратных средствах.

Архитектура компьютера — логическая организация, структура и ресурсы компьютера, которые может использовать программист. Архитектура определяет принципы действия, информационные связи и взаимное соединение основных логических узлов компьютера.

Архитектура клиент-сервер — способ организации взаимодействия программ или компонентов программы, подразумевающий наличие программы или компонента программы, называемого сервером, и одной или нескольких других программ или компонентов, называемых клиентами.

Ассемблер — алгоритмический язык низкого уровня, основной состав операторов которого базируется на машинных командах.

Ассемблер — транслятор с языка ассемблера в машинные команды.

Ассоциативная память — безадресная память. Поиск информации в которой производится по ее содержанию (ассоциативному признаку).

Атрибут (Attribute) — в базах данных — имя или структура поля записи. Атрибут характеризует размер или тип информации, содержащейся в поле.

Атрибут данных (Data attribute) — параметр данных, относящийся к их структурным свойствам, используемый для указания контекста данных или придания им смыслового значения.

Б

База данных (Database) — совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования. База данных (БД) является информационной моделью предметной области. Для создания и ведения базы данных (обновления информации и обеспечения доступа к ней пользователей) используется набор языковых и программных средств, называемый системой управления базой данных (СУБД).

Байт (b) — набор из стандартного числа (обычно 8) битов (двоичных единиц), используемый в качестве единицы количества информации при ее передаче, хранении и обработке на ЭВМ. В международных системах кодирования данных байт представляет код одного отображаемого (печатного) или управляющего символа.

Бегунок — небольшой прямоугольник на полосе прокрутки, используемый для перемещения по странице.

Бесконечный цикл — цикл, который не может завершиться нормальным образом.

Бета-версия (Beta version) — версия программного продукта, предшествующая выпуску коммерческого программного продукта. Предоставляется на льготных условиях с целью обкатки и выявления ошибок в новой системе.

Бета-тестирование — пробная эксплуатация программного продукта перед его выпуском на рынок.

Библиотека этапа выполнения (Run-time library) — библиотека процедур, обеспечивающих работу прикладной программы.

Библиотека пользователя — составленное программистом и определенным образом организованное личное собрание программ, подпрограмм, процедур, макроопределений, текстов и наборов данных.

Библиотека стандартных подпрограмм (Library) — совокупность подпрограмм, написанных на одном из языков программирования и удовлетворяющих единым требованиям к структуре, организации их входных и выходных данных и описаниям подпрограмм. Обычно библиотека хранится в виде файла во внешней памяти ЭВМ в рамках той или иной файловой системы, обеспечивающей автоматизированный доступ к отдельным алгоритмам и программам.

Бит — двоичная цифра, принимающая значения 0 или 1. Минимальная единица измерения количества передаваемой или хранимой информации.

Блок (Block) — в алгоритмических языках — программная единица, включающая описания объектов и исполняемые операторы (подпрограмма, функция).

Блок-схема (Flowchart) — в программировании — графическое представление программы или алгоритма с использованием стандартных графических элементов (прямоугольников, ромбов, трапеций и др.), обозначающих команды, действия, данные и т. п.

Буфер обмена — средство операционной системы Windows, предназначенное для обмена данными между приложениями.

В

Ввод данных — процесс записи данных в память компьютера с помощью устройства ввода.

Верификация программы — доказательство правильности программы, т. е. соответствия программы ее спецификациям.

Версия — вариант программного продукта или языка программирования.

Вертикальное меню — меню с вертикальным расположением пунктов (один под другим).

Ветвление программы — предусмотренная в программе возможность выполнения тех или иных действий в зависимости от результатов проверки некоторого условия.

Вещественное число — в информатике — тип данных, содержащий числа, записанные с десятичной точкой и/или с десятичным порядком.

Видеопамять — память, предназначенная для записи, хранения и считывания данных, определяющих изображение на экране дисплея.

Видеосистема компьютера — совокупность трех компонент: монитора, видеоадаптера и драйверов видеосистемы.

Виртуальная машина (Virtual machine) — условная вычислительная система, компоненты которой программным путем формируются или моделируются на основе ресурсов реального компьютера. На одном реальном компьютере может быть построено несколько виртуальных машин, каждая из которых выполняет свою программу и не препятствует работе другой виртуальной машины.

Виртуальная память (Virtual memory) — механизм управления памятью вычислительной системы, позволяющий программе использовать память, размер которой больше реальной оперативной памяти, имеющейся в компьютере. Физическое расположение виртуальной памяти на реальных носителях может не совпадать с логической адресацией данных в прикладной программе. Преобразование логических адресов программы в физические адреса запоминающих устройств обеспечивается аппаратными средствами и операционной системой.

Вложенный цикл — цикл, содержащийся в теле другого цикла.

Внедренный объект — объект, созданный средствами одного приложения и помещенный в документ другого приложения. Пример: диаграмма Microsoft Excel, вставленная в документ Microsoft Word.

Внешнее устройство ЭВМ — периферийное устройство для внешней обработки информации, в отличие от преобразований информации, выполняемых в ЭВМ.

Внешняя память — память, к содержимому которой можно обратиться только при помощи операций ввода-вывода. Реализуется при помощи внешних запоминающих устройств.

Воспроизведение программы или базы данных — согласно законодательству РФ — изготовление одного или более экземпляров программы для ЭВМ или базы данных в любой материальной форме, а также их запись в память ЭВМ.

Входные данные — данные, вводимые в вычислительную систему через устройства ввода для обработки или хранения.

Выходные данные — данные, поступающие из ЭВМ на устройства вывода в результате выполнения программы.

Выражение — конструкция на языке программирования, предназначенная для выполнения вычислений. Выражение может состоять из констант, переменных, указателей функций, объединенных знаками операций. Различают арифметические, логические и символьные выражения.

Выравнивание (Alignment) — способ размещения текста на странице: правый прижим, левый прижим, центрирование.

Г

Главная программа, основная программа — программа, выполняемая первой и управляющая вызовом подпрограмм.

Глобальная переменная (Global variable) — переменная, описание которой вынесено за пределы текущего блока программы. Глобальные переменные описываются в блоке, объемлющем текущий блок.

Графическая информация — сведения или данные, представленные в виде схем, эскизов, изображений, графиков, диаграмм, символов.

Графический адаптер — адаптер дисплея, поддерживающий вывод графической информации.

Графический режим — режим работы адаптера дисплея, обеспечивающий вывод графической информации.

Графический интерфейс пользователя — интерфейс пользователя, основанный на средствах машинной графики. Он предполагает взаимодействие человека с компьютером в форме диалога и использованием ввода и вывода на экран дисплея графической информации, управления программами с помощью кнопок, меню, окон, экранных панелей и других элементов управления, а также выделением цветом определенных частей изображения или текста и т. п. В графическом интерфейсе пользователя используются мышь, световое перо, цифровой планшет графического ввода и другие средства ведения диалога с компьютером и формирования графических изображений.

Графический объект — модель объекта реального мира, представленная в виде графического изображения.

Д

Данные (Data) — 1. Сведения, полученные путем измерения, наблюдения, логических или арифметических операций, представленные в форме, пригодной для постоянного хранения, передачи и автоматизированной обработки. 2. Информация, подготовленная для печати, хранения и обработки в вычислительной машине, т. е. представленная в символьной (цифровой) форме. Примерами Д. являются закодированные для ввода или уже введенные в компьютер текст, речь, изображение, таблицы всевозможных величин и т. п. 3. В контексте отдельной программы или пакета программ слово «Д.» означает все обрабатываемые программой объекты, отличные от ее команд.

Двоичная система счисления — позиционная система счисления с основанием, равным двум. Широко применяется для внутреннего представления данных в ЭВМ.

Двоичное число — число, представленное в двоичной системе счисления.

Двоичное представление — представление информации в двоичном коде. Д. п. является основным способом представления данных в памяти ЭВМ.

Двоичный код — код, символами которого являются 0 и 1.

Двойной щелчок мышью — действие, заключающееся в двукратном быстром нажатии и отпускании кнопки неподвижной мыши.

Двумерный массив — матрица. Массив с двумя измерениями (строками и столбцами).

Декомпилирование программы (согласно законодательству РФ) — технический прием, включающий преобразование объектного кода в исходный текст в целях изучения структуры и кодирования программы для ЭВМ.

Демонстрационная программа — программа, демонстрирующая интерфейс пользователя с программным продуктом, либо возможности программного продукта.

Демонстрационная база данных — база данных небольшого объема, представляющая собой фрагмент реальной БД и предназначенная для демонстрации возможностей СУБД или исходной БД.

Дерево файлов — графическое изображение структуры каталогов, подкаталогов и файлов на диске, указывающая на расположение файлов в подкаталогах и каталогах и подкаталогов в каталогах.

Диалог — двусторонний обмен информацией и управляющими сигналами между человеком и компьютером в форме вопросов и ответов в темпе, удобном для человека. Пользователь ведет Д. с помощью клавиатуры, мыши или микрофона.

Диалоговая система. — 1. Система программ и аппаратных средств, обеспечивающая диалоговый режим взаимодействия пользователя с исполняемой программой. 2. Программная система, управляемая пользователем в диалоговом режиме.

Диалоговый режим (Interactive mode) — способ взаимодействия пользователя или оператора с ЭВМ, при котором происходит непосредственный и двусторонний обмен информацией, командами или инструкциями между человеком и ЭВМ. Диалоговый режим подразумевает такую скорость обработки данных, которая не сказывается на технологии действий пользователя. Различают активные и пассивные диалоговые режимы.

Динамическая область памяти — область оперативной памяти, предназначенная для размещения прикладных программ на время их выполнения. После завершения программы Д. о. п. освобождается для загрузки другой программы.

Динамическое распределение памяти (Dynamic allocation) — управление ресурсами памяти, предполагающее выделение и освобождение памяти во время выполнения программы.

Директива — предложение или команда, управляющие вычислительной системой или ее компонентами. Д. содержит указания системе о необходимости выполнения определенных действий. Д. могут вводиться оператором ЭВМ или пользователем, а также содержаться в программе.

Дистрибутив — программные продукты в виде, поставляемом производителем (чаще всего на компакт-дисках). Дистрибутивный диск, как правило, содержит саму программу и инсталлятор для установки программы на жестком диске и настройки ее параметров. Иногда применяются дистрибутивные дискеты.

Длина — 1. Размер области памяти, отводимой для хранения объекта (программы, блока данных, записи, машинного слова и т. д.). Измеряется в битах, байтах и т. п. 2. Число элементов, образующих объект. Например, длина блока данных — это и размер в байтах области памяти, отведенной блоку данных, и число машинных слов в блоке; длина файла — это и количество байтов, занимаемых файлом в памяти, и число блоков в файле; длина дорожки — это и информационная емкость дорожки диска, и число размещенных на ней секторов.

Документация к программному продукту, программная документация — совокупность документов, описывающих назначение, структуру и применение программного продукта. Д. предназначена для облегчения использования программного продукта и включает как печатные документы (описание, руководства, учебники, справочники и т. п.), так и электронные документы (выводимые на экран и распечатываемые тексты, а также обучающие программы, средства оперативной справки и т. п.)

Документографическая база данных — база данных, содержащая библиографические описания документов и/или их рефераты.

Доступ — 1. Процедура установления связи с запоминающим устройством и размещенным на нем файлом для чтения, записи или изменения данных. 2. Возможность считать, записать или изменить данные. Обычно в этом случае указывается особенность или способ Д.

Драйвер устройства (Device driver) — программа, написанная специально для конкретного периферийного устройства с целью обеспечить управление этим устройством со стороны операционной системы.

Драйвер-русификатор — драйвер, поддерживающий ввод в память компьютера и вывод на экран дисплея букв русского алфавита (кириллицы). Обычно является резидентной программой, которая активизируется нажатием либо специальной клавиши, либо сочетания клавиш, служащего командой смены латинского алфавита на русский. При работе в текстовом режиме в знакогенератор адаптера дисплея загружаются изображения соответствующих клавиш клавиатуры.

Дружественный интерфейс — интерфейс, обеспечивающий пользователю не требующее специального обучения максимально удобное взаимодействие

с программой или вычислительной системой. Это наглядные, простые и понятные для него изображения на экране, значки, кнопки, меню, подсказки, звуковое сопровождение и т. п.

Ж

Жизненный цикл программы — совокупность взаимосвязанных и последовательных этапов существования программы. Согласно стандарту ISO/IEC 12207, существуют следующие этапы Ж. ц. п.:

- анализ предметной области и формулировка требований к программе;
- проектирование структуры программы;
- реализация программы в кодах;
- сопровождение программы;
- отказ от использования программы.

З

Завершение программы — последняя фаза обработки программы операционной системой, в ходе которой освобождаются использовавшиеся программой ресурсы, выводятся результаты, очищается выделенная программе оперативная память. З. п. может быть нормальным (после успешного выполнения программы) и аварийным.

Зависание — 1. Состояние вычислительной системы, при котором она перестает выдавать результаты и реагировать на запросы извне, например, на нажатие пользователем IBM PC клавиш Ctrl и Break, или на другие внешние прерывания. З. может быть вызвано аппаратным сбоем, ошибками в программном обеспечении или попыткой обращения прикладной программы к устройству, не подключенному к центральному процессору. 2. Состояние, характеризующееся аномально большим временем ожидания задачи или процесса при обращении к какому-либо ресурсу.

Загрузка — считывание данных из одного запоминающего устройства в другое, обычно обладающее большей оперативностью. Чаще всего этот термин обозначает копирование программ и данных из внешнего запоминающего устройства в оперативную память.

Загрузка программы — считывание программы из внешнего запоминающего устройства в оперативную память, настройка адресов и, возможно, запуск. З. п. выполняется под управлением загрузчика — специальной программы, обычно входящей в состав операционной системы.

Загрузочный модуль — программный модуль в виде, пригодном для загрузки и выполнения. З. м. получается из объектного модуля после редактирования связей и представляет собой программу в виде последовательности машинных команд, имеющую принятый для данной операционной системы формат.

Задание — совокупность программ и их данных, обрабатываемая операционной системой как единое целое. Содержание З. определяется пользователем в виде последовательности управляющих операторов, определяющих выполняемые программы и используемые ими данные.

Задача — 1. Задание вычислительной системы, представленное в виде программы или части программы и данных, которое операционная система рассматривает как единое целое при распределении ресурсов. В ряде операционных систем понятие «З.» совпадает с понятием «процесс», в других — с понятием «задание». 2. Процесс выполнения этого задания.

Закомментировать — временно отменить действие одного или нескольких операторов программы, превратив их в комментарий. Для этого достаточно ограничить эти операторы в соответствии с синтаксисом комментария, предусмотренным используемым языком программирования. Например, в языке Pascal нужно заключить оператор или последовательность операторов в скобки {...} или (*...*). Таким приемом часто пользуются для локализации ошибки при отладке программы. Чтобы восстановить первоначальный вид программы, достаточно убрать ограничители.

Закраска (в компьютерной графике) — изменение цвета пикселей, расположенных внутри изображенного на экране графического объекта.

Заккрытие файла — операция завершения работы программы с файлом. При ее выполнении все связанные с файлом области буферной памяти очищаются, а информация о произведенных изменениях данных заносится во внешнюю память (обычно на диск).

Заливка (в компьютерной графике) — однотонная закраска изображенного на экране плоского графического объекта, ограниченного четким контуром.

Запись — 1. Процесс запоминания (фиксирования) данных в запоминающей среде или на носителе данных (writing). 2. Результат процесса записи (record). 3. Единица обмена данными между программой и внешней памятью, т. е. совокупность данных, совместно пересылаемых на периферийные устройства или с периферийных устройств одним оператором ввода-вывода. Поэтому файлы, хранимые во внешнем ЗУ, часто представляют собой набор З.

Запрос — обращение с целью получения ответа. Это может быть сигнал, с помощью которого операционная система запрашивает периферийные устройства об их готовности к обмену данными, либо обращение пользователя к базе данных, содержащее задание на поиск, чтение и обработку нужной информации, либо обращение программы к устройству ввода с целью получения данных.

Запуск программы — загрузка программы в оперативную память и передача управления команде программы, выполняемой первой.

Зарезервированное слово (в языках программирования) — слово, смысл которого зафиксирован правилами языка и по которому транслятор (или человек) распознает основные языковые конструкции. Не разрешается их использова-

ние в качестве идентификаторов, поскольку они являются зарезервированными (служебными) словами языка.

Знак — отдельный символ алфавита или элемент системы символов, используемой в вычислительной системе.

Знак арифметической операции — знак, определяющий арифметическую операцию. Как правило, это знаки «+», «-», «*», «/».

Значение — смысл или величина, содержащаяся в элементе данных.

Значение по умолчанию — стандартное значение, присваиваемое параметру автоматически, если пользователь не задаст ему иное значение.

Значок — условное изображение программы, операции или информационного объекта на экране. Обычно З. отображаются на кнопках или обозначают пункты меню.

И

Идентификатор — символическое имя переменной или подпрограммы, которые однозначно идентифицируют их в программе.

Изменение размера окна — многие виды окон допускают изменение своих размеров пользователем. Чтобы изменить размер окна, нужно подвести указатель мыши к границе окна так, чтобы он принял соответствующий вид, нажать левую кнопку мыши и перетащить мышью границу окна в нужном направлении.

Импорт — операция переноса данных из одной системы, программы или документа в другую систему, программу или документ.

Индекс — число целого типа или арифметическое выражение, принимающее целочисленное значение, приписываемое элементу массива или другой конструкции данных, для идентификации этого элемента.

Индикатор — 1. Элемент данных, отражающий текущее состояние устройства или данных. Например, поле данных, в которое системная или прикладная программа помещает код ошибки, называют индикатором ошибки. 2. Элемент графического интерфейса пользователя, показывающий ход выполнения какого-либо процесса.

Инициализация — приведение программы или устройства в состояние готовности к использованию. И. программы заключается в задании начальных значений или программных переменных (адресов, счетчиков, переключателей, указателей и т. п.) перед выполнением программы.

Инсталляция, установка — установка программного продукта на компьютер. И. обычно выполняется под управлением инсталлятора — программы, которая приводит состав и структуру устанавливаемого программного изделия в соответствие с конфигурацией компьютера, а также настраивает программные параметры согласно типу имеющейся операционной системы, классам

решаемых задач и режимам работы. Таким образом, И. делает программный продукт пригодным для использования в данной вычислительной системе и готовым решать определенный класс задач в определенном режиме работы.

Инструкция — 1. В некоторых языках программирования — то же, что команда, предложение или оператор. 2. Документ, указывающий и определяющий порядок работы при установке и эксплуатации программного продукта.

Инструментальные средства — программные средства (сервисные программы, пакеты прикладных программ, оболочки), предназначенные для автоматизации создания, редактирования и отладки различных программных продуктов: программ, баз данных и знаний, электронных таблиц, текстов, графических изображений и т. д. Например, текстовый редактор, позволяющий набирать текст программы, отладчик, облегчающий программисту выполнение отладки разрабатываемой программы, библиотека программ, облегчающая разработку пользовательского интерфейса программы и т. д.

Интегрированная среда — система, включающая все необходимые пользователю программные средства и обеспечивающая единообразное взаимодействие с ними. Примером И. с. может служить среда разработки программ Delphi вместе с набором приложений, обеспечивающим потребности программистов при создании программ.

Интерпретатор — транслятор (программа или устройство), анализирующий команды или операторы исходной программы, написанной на алгоритмическом языке высокого уровня, и немедленно выполняющий их. Таким образом, И. одновременно и транслирует и выполняет заданную программу, делая это покомандно или пооператорно.

Интерпретация — метод выполнения программы, при котором каждый отдельно взятый оператор транслируется и сразу выполняется, после чего осуществляется переход к следующему оператору. И. выполняется с помощью программных или аппаратных средств, называемых интерпретаторами. Достоинством И. является возможность пошагового прослеживания и выполнения программы, недостатком — относительно малая скорость выполнения.

Интерфейс (Interface) — программное или аппаратное обеспечение коммуникации между компьютером и его пользователем или между двумя устройствами.

Интерфейс пользователя (User interface) — интерфейс, определяющий процессы взаимодействия пользователя с информационным ресурсом.

Информация (Information) — согласно законодательству РФ — сведения о лицах, предметах, фактах, событиях, явлениях и процессах независимо от формы их представления. Информация уменьшает степень неопределенности, неполноту знаний о лицах, предметах, событиях и т. д.

Информационный продукт (Information product) — документированная информация, подготовленная в соответствии с потребностями пользователей

и представленная в форме товара. Информационными продуктами являются программные продукты, базы и банки данных и другая информация.

Исключение — прием программирования, используемый при отладке программ и для повышения надежности программ. Если программисту известно, что при правильной работе программы должны выполняться некоторые условия, то невыполнение этих условий свидетельствует о неправильной работе программы.

Исполняемый оператор — оператор в программе, которому соответствует одна или несколько последовательных операций, составляющих алгоритм решения задачи. Например, оператор присваивания или оператор цикла.

Исполняемый файл — файл, готовый к выполнению операционной системой. В IBM PC И. ф. имеют расширения exe, com и bat. Файлы с расширениями exe и com — это программы. Файлы с расширением bat — командные файлы.

Испытание программы, тестирование программы — проверка программы или ее составной части путем реального выполнения специально подобранных контрольных задач. И. п. является важным этапом разработки программы и представляет собой проведение испытательных прогонов программы с целью убедиться, что она действительно решает ту задачу, для которой предназначена, и выдает правильный ответ при любых условиях.

Исходные данные — данные, необходимые для решения задачи. И. д. подготавливаются заранее и используются программой в ходе ее выполнения. И. д. вводятся при помощи операторов ввода программы или пользователем в диалоговом режиме. При этом они становятся входными данными. И. д. для работы подпрограммы передаются ей в основном через механизм формальных и фактических параметров.

Исходный код — текст программы на алгоритмическом языке. В компьютере исходный текст либо непосредственно выполняется интерпретатором, либо предварительно переводится компилятором в стандартный загрузочный код, способный многократно исполняться в определенной вычислительной среде.

Итерационный цикл — оператор цикла, для которого число повторений тела цикла заранее неизвестно. В итерационных циклах на каждом шаге вычисления происходит последовательное приближение и проверка условия достижения искомого результата. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.

К

Каталог (Папка) — в широком смысле — перечень (оглавление) наборов данных, расположенных на внешнем носителе. В оглавлении обычно указываются имя набора данных, его размер, дата создания и местоположение.

Каталог — оглавление файлов, доступное пользователю посредством интерпретатора команд операционной системы. Обычно система файловых каталогов имеет иерархическую структуру.

Килобайт (Кб, КБайт) — единица измерения объема информации, численно равная 1024 байт.

Клавиша оперативного вызова, «горячие» клавиши — клавиша или сочетание нескольких клавиш, одновременное нажатие на которые вызывает определенные действия программы или вычислительной системы.

Ключевое слово — слово или сочетание слов естественного языка, отражающее содержание текста.

Ключ — элемент данных, выражающий местоположение записи или группы в файле.

Кнопка (в интерактивных системах) — элемент управления графического интерфейса пользователя, выбор которого с помощью указателя мыши вызывает определенное действие системы.

Код — 1. Система символов и однозначных правил их интерпретации, используемая для представления информации в виде данных. В современных компьютерах широкое распространение получил двоичный К. В нем каждая буква, цифра, знак препинания, пробел или какой-либо другой символ однозначно представляются (кодируются) комбинацией двоичных цифр — нулей и единиц. 2. Программа, записанная на языке программирования. В таком смысле слово «К.» применяется, чтобы подчеркнуть отличие готовой программы от ее блок-схемы, или наброска, выполненного на некотором условном языке.

Кодирование символов — представление набора символов в виде последовательности цифр. Наиболее часто используемыми символами являются буквы, знаки, цифры и т. п., которые представляются двоичными числами.

Колонка, столбец — вертикальный ряд объектов одной природы. Обычно К. образуют элементы данных или их позиции, расположенные на экране, бумаге или другом носителе вертикально — друг под другом. Например, столбец элементов матрицы, К. ячеек электронной таблицы.

Машинная команда — описание элементарной операции, которую должен выполнить компьютер. Команды хранятся в ячейках памяти в двоичном коде. В общем случае команда содержит код выполняемой операции, указания по определению операндов (или их адресов) и указания по размещению получаемого результата. Результат выполнения команды вырабатывается по точно определенным правилам, заложенным в конструкцию компьютера. В зависимости от количества операндов различают одноадресные, двухадресные, трехадресные и переменнo-адресные команды.

Команда ввода — машинная команда, определяющая ввод данных с внешнего устройства в основную память.

Команда возврата — команда перехода, предписывающая выход из подпрограммы и возврат в вызывающую программу.

Команда вывода — машинная команда, определяющая вывод данных из основной памяти во внешнее устройство.

Команда операционной системы — предписание, инициирующее выполнение операционной системой определенных действий, например, создать новый каталог или напечатать файл. К. о. с., как правило, состоит из имени команды и, возможно, параметров, содержащих необходимую уточняющую информацию.

Команда паузы — команда, предписывающая временное прекращение выполнения программы.

Команда прерывания — команда, вызывающая прерывание.

Командная строка — строка экрана дисплея, предназначенная для записи вводимых пользователем команд операционной системы и запуска программ с клавиатуры.

Комментарий — ограниченный специальным образом фрагмент программы, предназначенный для чтения человеком и игнорируемый при трансляции. К. служит для включения в текст программы пояснений, облегчающих человеку ее чтение и анализ. В каждом языке программирования имеется свой синтаксис К.

Коммерческий программный продукт — программный продукт, изготовленный для продажи.

Компилятор (программа) — транслятор, выполняющая компиляцию программных модулей. В отличие от интерпретатора, К. только преобразует программу, составленную на языке программирования высокого уровня в программу на машинном языке или языке, близком к машинному, не участвуя в ее исполнении. На вход К. поступает исходный модуль, который после компиляции преобразуется в объектный модуль.

Компиляция — трансляция программы или отдельного программного модуля, написанного на языке программирования высокого уровня (исходная программа, исходный модуль) в программу или модуль на машинном языке или языке, близком к машинному (объектная программа, объектный модуль). В процессе К. программа преобразуется в промежуточную форму, к которой впоследствии необходимо присоединить библиотечные средства, содержащие стандартные подпрограммы и процедуры, а если нужно, то можно добавить любые другие модули, написанные самим пользователем, скомпилированные в объектные модули, возможно, с иных языков высокого уровня.

Компонент программы — часть программы, которая структурно отделена от остальных частей и может использоваться в нескольких программах.

Компоновка — процесс сборки загрузочного модуля из полученных в результате отдельной компиляции объектных модулей с автоматическим поиском и присоединением библиотечных подпрограмм и процедур. В процессе К. программа собирается в единое целое.

Компьютер — программируемое электронное устройство, способное обрабатывать данные и производить вычисления, а также выполнять другие задачи манипулирования символами.

Конвертор — программа для перекодирования данных из одного машинного кода в другой или из одного формата в другой.

Конец файла — символ, указывающий конец текстового файла.

Константа — элемент данных, присутствующий в тексте программы и не меняющий своего значения при многократном ее использовании.

Контекстное меню — меню, которое содержит команды, применимые в данном контексте. Например, во многих приложениях Windows, если выделить какой-либо объект и щелкнуть правой кнопкой мыши, то появится К. м., содержащее команды, применимые к выделенному объекту в его текущем состоянии.

Контекстно-зависимая справка — справочная система, которая выдает пользователю справочную информацию с учетом текущего контекста его работы. Например, если выделен некоторый объект, то при нажатии клавиши F1 выводится список разделов справки, имеющих отношение к работе с объектами данного типа.

Л

Линейный алгоритм — набор инструкций, выполняемых последовательно.

Листинг — печатный документ, формируемый вычислительной системой и содержащий данные о ходе и результатах разных этапов выполнения задачи. Например, Л., формируемый на этапе трансляции, обычно содержит текст исходной программы и результаты трансляции (сведения об ошибках или характеристики полученного объектного модуля).

Логическая переменная — переменная, принимающая только логические значения: «истина» или «ложь», которые в ЭВМ могут быть представлены в виде 1 и 0. В языках программирования эти значения обычно обозначаются 1, true, T или 0, false, F, соответственно. Л. п. вводятся в программу путем описания переменной, в котором указываются идентификатор (имя) переменной и ключевое слово, определяющее логический тип. В качестве ключевых слов в описаниях Л. п. применяются: logical — в Фортране и Boolean — в Pascal.

Логическая запись — совокупность записей взаимосвязанных элементов данных, рассматриваемая в логическом плане как единое целое. Одна логическая запись может состоять из нескольких физических или быть частью одной физической записи.

Логический диск — область хранения информации на жестком диске, обозначаемая латинскими буквами C, B, E и т. д. Разбиение на логические диски условно и служит для удобства работы с жестким диском.

Логический тип (Булевский тип) — тип данных, принимающих только логические значения: «истина» или «ложь», над которыми производятся логические операции.

Логическое выражение (булево выражение) — совокупность одной или нескольких логических переменных, логических функций и отношений, соединенных знаками логических операций и скобками.

Локальная база данных — база данных, размещенная на одном или нескольких носителях на одном компьютере.

Локальная переменная — переменная с ограниченной областью определения в программе. Л. п. существует только в том модуле или блоке, где она описана, а вне его является недоступной.

М

Макрокоманда (Макрос) — 1. Выражение в тексте программы, вместо которого подставляется текст, заданный макроопределением. Макроопределения часто используются в макроассемблерах. 2. Последовательность команд, запускаемая одним нажатием клавиши на клавиатуре или кнопки на экране дисплея.

Маска — шаблонная комбинация разрядов или символов, используемая для анализа данных. Обычно с помощью М. проводятся выделения, сравнения или исключения отдельных частей данных.

Массив — именованная последовательность однотипных элементов, число которых фиксировано. Каждый элемент М. однозначно определяется именем М. и набором целых чисел, называемых индексами М.

Машинный язык — совокупность машинных команд, отличающихся количеством операндов в команде, назначением информации, задаваемой в операндах, и набором операций, которые может выполнить компьютер.

Машинная команда — команда, которая может быть распознана и выполнена центральным процессором, т. е. команда, входящая в систему команд конкретной ЭВМ.

Машинная программа — программа на машинном языке. М. п. является последовательностью машинных команд и может быть распознана и выполнена центральным процессором.

Машинное время — время, затрачиваемое ЭВМ на выполнение определенного комплекса вычислений.

Машинное слово — последовательность битов или символов определенной длины, воспринимаемая памятью, арифметическим устройством или устройством управления ЭВМ как единое целое, имеющее определенное содержание.

Машинный сбой (сбой) — кратковременный отказ или ошибка в работе аппаратных средств вычислительной системы.

Мегабайт (Мбайт, Мб, М) — единица количества информации; 1 Мб = = 1024 Кб = 13 048 576 байт.

Меню (Мени) — изображаемый на экране дисплея список режимов, команд или вариантов ответа, предлагаемых пользователю для выбора. Предлагае-

мые варианты называются пунктами М. Они определяют последующие действия системы. Если один из пунктов М. выделен другим цветом (цветовым маркером), значит М. активно. Меню является одним из основных элементов графического интерфейса пользователя и одним из средств реализации интерактивного режима взаимодействия пользователя с вычислительной системой.

Метка — 1. Целое число без знака или идентификатор, приписанный оператору программы и используемый в других частях программы для обращения к этому оператору. М. обычно ставится перед оператором и отделяется от него двоеточием, как в Pascal и C. Оператор, снабженный меткой, называют помеченным оператором. 2. Физическая запись на внешнем носителе данных, определяющая начало или конец файла или тома.

Модификация программы или базы данных (согласно законодательству РФ) — любые изменения программы для ЭВМ или базы данных, не являющиеся адаптацией.

Мультимедиа — программные и аппаратные средства, обеспечивающие воспроизведение видеoinформации с соответствующим звуковым сопровождением.

Мусор — ненужные, не подлежащие дальнейшему использованию данные в памяти ЭВМ. М. напрасно занимает место в памяти, поэтому ее следует регулярно очищать от М. Эту процедуру называют «сборкой мусора».

Н

Навигатор — программа поиска записей в базе данных или в информационных массивах.

Наследование — понятие объектно-ориентированного программирования, которое состоит в том, что класс, определяемый на основе другого класса, наследует все или некоторые свойства и методы родительского класса.

Неактивная программа — загруженная в оперативную память программа, которая в текущий момент не управляет центральным процессором. Например, ожидающая обращения резидентная программа.

Неактивное окно — окно, не используемое пользователем в настоящий момент. Н. о. может быть полностью или частично перекрыто другими окнами. Чтобы сделать его активным, как правило, достаточно навести на него указатель мыши и нажать левую кнопку.

О

Оболочка (Shell) — программа, создаваемая для упрощения работы со сложными программными системами. Оболочки преобразуют неудобный командный пользовательский интерфейс в дружелюбный графический интерфейс или интерфейс типа меню. Обычно оболочка реализуется в виде отдельной программы

Обработка данных — выполнение по заданной программе определенных действий над данными.

Обработка данных в реальном масштабе времени — обработка данных, протекающая с такой же скоростью, что и моделируемые события.

Обрабатывающая программа — программа, обрабатывающая данные или другие программы. Например, текстовый редактор, компилятор, редактор связей, загрузчик.

Обработка прерываний — стандартные действия, выполняемые операционной системой или аппаратными средствами при возникновении прерываний.

Объект — предмет или явление, которому можно присвоить название.

Объектно-ориентированное программирование (ООП) — наиболее популярная в настоящее время методология программирования, являющаяся развитием структурного программирования. Центральной идеей ООП является инкапсуляция, т. е. структурирование программы на модули особого вида, объединяющие данные и процедуры их обработки, причем внутренние данные модуля могут быть обработаны только предусмотренными для этого процедурами.

Объектный модуль — программный модуль, являющийся результатом компиляции исходного модуля. О. м. представляет собой последовательность машинных команд, готовую к объединению с другими О. м. с помощью редактора связей (компоновщика).

Объектографическая база данных — фактографическая база данных, содержащая расширенный набор данных о сложных объектах предметной области.

Одномерный массив — массив с одним индексом.

Окно — выделенная часть экрана дисплея, с которой программа или пользователь работает как с отдельным независимым экраном, размеры и расположение которого пользователь может изменять. Различают несколько типов окон: О. приложения, О. документа, диалоговое О.

Оператор — Допустимое в языке программирования высокого уровня предложение, задающее целостное законченное действие ЭВМ, или представляющее набор описаний. Типичными О. в традиционных языках программирования являются О. ввода-вывода, присваивания, перехода, цикла, процедуры и др. Грамматическая конструкция каждого из них определяется синтаксисом конкретного языка программирования.

Оператор ввода — оператор в программе, предписывающий передачу данных из устройства ввода или внешней памяти в оперативную память и делающий эти данные доступными программе.

Оператор выбора, переключатель — оператор в программе, определяющий выбор одной из нескольких ветвей алгоритма. Например, с помощью О. в. программируется меню (выбор пользователем последовательности действий в соответствии с пунктом меню).

Оператор вывода — оператор в программе, предписывающий передачу данных из оперативной памяти во внешнюю память или на устройство вывода.

Оператор присваивания — оператор, передающий значение арифметического, логического или другого выражения одной или нескольким переменным, либо имени функций.

Оператор цикла — оператор, упрощающий программирование цикла. О. ц. формально состоит из заголовка цикла и тела цикла. Заголовок цикла указывает на последовательность повторяемых операторов, образующих тело цикла, и либо определяет множество значений параметра цикла и предписывает многократное выполнение тела цикла при этих значениях параметра, либо определяет условие повторного выполнения тела цикла.

Операция — действие, выполняемое над данными.

Операционная система (ОС) — комплекс программ, обеспечивающий выполнение других программ, распределение ресурсов, планирование, ввод-вывод данных, управление данными, взаимодействие с оператором.

Описание — раздел программы, идентифицирующий структуры данных, которыми должна манипулировать программа, и описывающий их типы.

Описание алгоритмического языка — документ, специфицирующий возможности алгоритмического языка. Обычно описание содержит алфавит допустимых символов и ключевых слов, синтаксические правила построения из алфавита допустимых конструкций языка и семантику, объясняющую смысл и назначение конструкций языка.

Опция — элемент меню; один из предлагаемых вариантов выбора.

Останов — прекращение автоматической выборки и выполнения команд центральным процессором. О. может быть вызван причинами, связанными с выходом из строя одного из устройств ЭВМ или операционной системы — аварийный О., а может быть программируемым — предусмотренным выполняемой программой.

Открытие файла — операция начала работы программы с файлом. «Открыть файл» означает связать программу с файлом: установить соответствие между физическим файлом на внешнем запоминающем устройстве, буферной памятью и набором данных программы, заполнить соответствующие поля в управляющих таблицах операционной системы и т. д.

Открытый файл — файл, для которого выполнена операция открытия файла, в результате чего его записи стали доступными для чтения и обработки.

Отладка программы (Debugging) — этап разработки программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

Отладчик (Debugger) — программа, позволяющая исследовать внутреннее устройство программы. Отладчик обеспечивает пошаговое выполнение программы, просмотр текущих значений переменных, вычисление значения любого выражения в программе и другие функции.

П

Пакет прикладных программ — комплект программ, предназначенных для решения задач из определенной проблемной области. Обычно применение пакета прикладных программ предполагает наличие специальной документации: лицензионного свидетельства, паспорта, инструкции пользователя и т. п.

Пакетная обработка (Пакетный режим) — обработка данных или выполнение заранее подготовленных заданий без участия пользователя.

Палитра — 1. Множество цветов, которые можно отобразить на экране дисплея. Монитор и видеоадаптер компьютера определяют разрешающую способность экрана и максимально возможное количество отображаемых на нем цветов. Однако имеющееся программное обеспечение может не использовать всех графических возможностей, предоставляемых аппаратурой, и ограничить число цветов, доступных для формирования изображения. Цвет каждого пиксела в машинной программе задается двоичным кодом. Поэтому количество цветов в П. определяется числом разрядов, отведенным для представления цвета одного пиксела. Например, если для представления цвета пиксела отводится 4 разряда, то пиксел может иметь 16 цветов (коды от 0000 до 1111). Следовательно, 4-разрядный пиксел допускает шестнадцатичетную П. 2. Таблица соответствия между кодами цветов в программах рисования или набор цветов на экране дисплея. 3. Набор цветов в программах рисования или набор цветов и инструментов рисования в программах раскраски.

Панель инструментов (инструментальная панель) — панель экрана с размещенным на ней графическим меню.

Панель экрана (экранная панель, панель) — выделенная часть экрана дисплея или окна, предназначенная для размещения меню, управляющих кнопок, справочной и другой информации.

Параметр цикла (переменная цикла, управляющая переменная цикла) — переменная, используемая в операторе цикла для управления повторением тела цикла.

Пассивный диалог — режим взаимодействия пользователя и программной системы, инициатива ведения которого принадлежит программной системе. При этом программная система ведет за собой пользователя, запрашивая в точках ветвления вычислительного процесса дополнительную информацию, необходимую для принятия заложенных в алгоритм решений. В пассивном диалоге программная система обеспечивает пользователя информационными сообщениями и подсказками, облегчающими использование диалоговой системы. Запросы к пользователю строятся обычно либо в виде меню, либо в виде шаблонов.

Паскаль — язык программирования высокого уровня. Первая версия П. разработана швейцарским ученым Н. Виртом, была опубликована в 1968 г. и предназначалась для обучения программированию, как систематической дисциплине.

Переменная (Variable) — именованный элемент данных в программе. П. различаются по имени и могут принимать разные значения. Значение П. может быть получено и изменено программой. Тип данных, к которому могут принадлежать значения П., устанавливается описанием переменной.

Переносимость программы (мобильность программы) — свойство программы, позволяющее переносить ее без изменений или с минимальными модификациями с одного компьютера на другой, в особенности когда эти ЭВМ имеют различную архитектуру.

Переполнение — ситуация, при которой результат операции превышает по абсолютной величине наибольшую представимую в вычислительной системе величину. Обычно П. приводит к аварийному завершению задачи.

Внешние устройства (периферийные устройства) — часть технического обеспечения, конструктивно отделенная от основного блока компьютера.

Периферийные устройства — комплекс устройств для внешней обработки данных, обеспечивающий их подготовку, ввод, хранение, управление, защиту, вывод и передачу на расстояние по каналам связи.

Пиксел — элемент графической информации. Минимальный независимый участок изображения, для которого можно задать цвет, яркость и другие характеристики. П. называются точки, на которые делится экран дисплея при графическом режиме работы или точки, из которых формируется изображение на струйных или лазерных принтерах.

Графическое меню — меню, пункты которого изображаются на экране дисплея в виде кнопок с нарисованными на них значками.

Платформа (Platform) — в зависимости от контекста — либо комбинация аппаратуры и операционной системы, либо аппаратура, включая тип процессора.

Подменю — меню, вызываемое выбором пункта меню вышележащего уровня. Обычно пункты меню, в результате выбора которых раскрывается П., помечаются многоточием или стилизованной стрелкой.

Подпрограмма — выделенная часть программы, реализующая определенный алгоритм и допускающая обращение из разных мест остальной части программы. Применение П. сокращает текст программы, если на разных этапах решения задачи требуется выполнить один и тот же алгоритм, так как при этом исключается необходимость многократного повторения в программе одних и тех же групп команд или операторов.

Поле данных (Поле, Field) — часть записи или заполняемой формы, имеющая функционально самостоятельное значение и обрабатываемая как отдельный элемент данных.

Полное имя файла — составное имя, включающее последовательно записываемые через разделитель «\»: — имя диска, — последовательность имен вложенных каталогов, через которые проходит путь к файлу (начиная с высшего в их иерархии), и имя файла с расширением. Таким образом, П. и. ф. состоит из маршрута поиска файла и имени файла с расширением.

Полнотекстовая база данных — база данных, в которой хранятся записи полнотекстовых документов или их частей.

Полоса прокрутки — элемент управления, служащий для управления прокруткой изображения. Обычно это расположенная на границе окна полоса с бегунком, перемещая который при помощи мыши, можно переместить изображение или текст в окне по вертикали или по горизонтали.

Последовательность команд (Command sequence) — в алгоритмических языках — совокупность операторов или зарезервированных слов.

Последовательный доступ — 1. Способ доступа к данным, при котором записи файла считываются в том же порядке, в котором они были записаны при его создании. Такая логическая последовательность записей вовсе не означает, что и физически записи расположены последовательно, например, для файла, размещенного на магнитном диске, хотя логическая и физическая последовательности часто совпадают. 2. Способ доступа, при котором данные считываются в оперативную память в порядке их физического размещения на носителе данных внешнего запоминающего устройства. При этом для обращения к нужной записи необходимо последовательно «просмотреть» все участки носителя, начиная либо с самого первого, либо со следующего за тем, к которому было предыдущее обращение. Такой способ доступа осуществляется, например, к данным, хранящимся на магнитных лентах.

Прерывание — временное прекращение выполнения команд программы с сохранением информации о ее текущем состоянии и передаче управления специальной программе — обработчику прерываний.

Прерывание ввода-вывода — внешнее прерывание, связанное с работой устройства ввода-вывода.

Прикладная программа, Приложение (Application) — программа, реализующая обработку данных в определенной области применения.

Прикладное программное обеспечение — программное обеспечение, состоящее из отдельных прикладных программ и пакетов прикладных программ, предназначенных для решения различных задач пользователей и созданных на их основе автоматизированных систем.

Проблемно-ориентированная база данных — база данных, содержащая тематически связанные документы и/или данные, предназначенные для решения прикладных задач определенного вида.

Программа (Program) — последовательность машинных команд, предназначенная для достижения конкретного результата.

Программное обеспечение (Software) — совокупность входящих в состав вычислительной системы программных средств: программ, данных и документов. П. о. обеспечивает эффективную работу ЭВМ и предоставляет пользователю определенные виды обслуживания.

Программное прерывание — прерывание, вызванное либо обращением программы к аппаратным средствам (например, ввод исходных данных, вывод

результатов, очистка экрана и т. п.), либо в связи с некорректным представлением или использованием команд или данных.

Программный модуль — программа или часть программы, оформленная в виде, допускающем ее независимую трансляцию.

Программный продукт, программное изделие — программа (пакет программ), предназначенная для продажи или передачи в эксплуатацию другим лицам и удовлетворяющая ряду требований, важнейшие из которых: 1) сама программа и предлагаемая к ней инструкция должны содержать достаточное количество данных для своего полноценного использования; 2) программа должна сопровождаться производителем, т. е. замеченные ошибки должны устраняться бесплатно для покупателей; 3) программа должна поставляться в удобном для установки и использования виде, как правило, на гибких или лазерных дисках с инструкцией и защитной упаковкой; 4) программа должна быть изготовлена с помощью законно приобретенных программных средств и запатентована.

Программирование — процесс подготовки задач для их решения с помощью компьютера; процесс составления программ.

Программное обеспечение (Software) — комплекс программ, обеспечивающих обработку или передачу данных.

Прокрутка изображения — вертикальное или горизонтальное перемещение изображения в окне экрана. Управление П. осуществляется клавишами управления курсором или при помощи мыши путем перемещения бегунков на полосах прокрутки.

Процессор — устройство компьютера, выполняющее команды. Обязательными компонентами П. являются арифметико-логическое устройство и устройство управления. П. характеризуются архитектурой, набором выполняемых команд, скоростью их выполнения и длиной машинного слова. Архитектура определяет типы обрабатываемых данных, регистры, стеки и систему адресации. Набор команд может быть фиксированным или допускать возможность микропрограммирования.

Прямой доступ — способ доступа к данным, при котором все элементы данных или записи в файле являются равнодоступными, т. е. время доступа к элементу или записи не зависит от расположения данных в памяти и для доступа к любому из них не требуется «просмотр» других элементов или записей.

Псевдокод — система обозначений и правил, предназначенная для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками.

Пункт меню (Элемент меню) — один из вариантов, предлагаемых пользователю для выбора. Каждый П. м. обозначает какое-то действие или набор действий системы, поэтому выбирая тот или иной П. м., пользователь сам определяет последующие действия системы.

Путь к файлу — данные, указывающие операционной системе место в памяти, где следует искать файл. Обычно П. указывает логическое имя накопителя и последовательность имен вложенных каталогов, в последнем из которых содержится нужный файл.



Рабочий лист — формализованная анкета, предназначенная для обработки и записи структурированных данных. Рабочий лист содержит состав полей данных, соответствующих виду обрабатываемых документов или данных, а также набор сведений об их содержании и правилах заполнения.

Размерность массива — количество индексов, необходимое для однозначной идентификации любого элемента массива.

Размер шрифта (кегель шрифта) — высота символов шрифта, измеряемая в пунктах (1 пункт = $1/72$ дюйма, или, примерно 0,36 мм). Наиболее ходовым размером шрифта считается 10 пунктов. Более мелкие шрифты (8 пунктов и менее) трудно читаются.

Раскрывающееся меню (Pull-down menu) — список команд меню, раскрывающийся из строки меню и остающийся доступным пользователю по мере необходимости.

Раскрывающийся список — элемент управления, предназначенный для выбора одного или нескольких значений из списка значений. Аналогичен элементу управления «список», но занимает меньше места на экране.

Распространение программы или базы данных (согласно законодательству РФ) — предоставление доступа к воспроизведенной в любой материальной форме программе или базе данных, в том числе сетевыми и иными способами, а также путем продажи, проката, сдачи внаем, предоставления взаймы, включая импорт для любой из этих целей.

Расширение файла — последовательность символов, предназначенных для идентификации типа файла. Обычно расширение состоит не более чем из трех символов, отделяемых от имени файла точкой.

Редактирование — внесение изменений в текст или преобразование программ или данных к виду, требуемому для их дальнейшего использования.

Резидентная программа — программа, постоянно присутствующая в оперативной памяти. После считывания в оперативную память Р. п. не выполняется до конца, а остается в состоянии готовности к продолжению выполнения до своей выгрузки, перезагрузки или выключения компьютера. Она активизируется и выполняет заданные действия либо при обращении к ней, например, нажатием определенной клавиши или сочетания клавиш, либо при достижении компьютером определенного состояния (например, по сигналу таймера).

Рекурсивная подпрограмма — подпрограмма, которая вызывает сама себя.

С

Семантика — система правил истолкования отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка.

Сервер (Server) — в широком смысле — объект, предоставляющий сервис другим объектам по их запросам.

Сервер — в информационных сетях — компьютер или программная система, предоставляющая удаленный доступ к своим службам или ресурсам с целью обмена информацией.

Сетка — совокупность вертикальных и горизонтальных координатных линий, на фоне которых размещаются какие-либо графические объекты.

Синтаксис языка программирования — совокупность правил написания чисел, переменных, выражений, операторов, процедур и других элементов и предложений (синтаксических конструкций) данного языка программирования.

Синтаксическая ошибка — ошибка в программе, связанная с нарушением синтаксиса языка программирования.

Система программирования — система для разработки программ на конкретном языке программирования. Система программирования предоставляет пользователю специальные средства разработки программ: транслятор, (специальный) редактор текстов программ, библиотеки стандартных подпрограмм, отладчик и др.

Система управления базами данных — комплекс программных и лингвистических средств общего или специального назначения, реализующий поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей в условиях принятой технологии обработки данных.

Системные программы — программы общего пользования, выполняемые вместе с прикладными программами и служащие для управления ресурсами компьютера: центральным процессором, памятью, вводом-выводом.

Слово — не содержащая пробелов последовательность символов некоторого алфавита, имеющая определенное смысловое значение.

Сообщение (Message) — 1. Информация о ходе или состоянии вычислительного процесса, выдаваемая пользователю компонентами вычислительной системы. 2. Порция данных, оформленная для передачи данных принятым в данной сети ЭВМ или системе компьютерной связи образом.

Сопровождение программного изделия — меры, направленные на поддержание в работоспособном состоянии находящейся в эксплуатации программы: устранение выявленных в ходе эксплуатации ошибок и внесение изменений в программу с целью улучшения ее функциональных возможностей в соответствии с изменением предъявляемых к ней требований.

Сортировка данных (сортировка, упорядочение) — распределение элементов данных по группам в соответствии с определенными правилами.

Сохранение — запись группы данных из оперативной памяти в файл, находящийся во внешней памяти.

Специальные клавиши — группа клавиш, предназначенных для управления работой компьютера или вызова стандартных действий.

Список — элемент управления, предназначенный для выбора одного или нескольких значений из их предопределенного набора.

Справка (справочная система) — программное средство, предназначенное для предоставления пользователю информации о некотором приложении и о порядке работы с ним в интерактивном режиме, т. е. непосредственно во время работы с приложением.

Среда программирования — интегрированная система разработки программ. В С. п. все программные средства имеют единый пользовательский интерфейс, общую базу данных и не требуют специального вызова, так что программист может выполнять свою работу, не выходя в операционную систему.

Ссылка — имя, указатель или адрес в программе, указывающий на объект программы, подпрограмму, другую программу или устройство.

Строка (строка символов) — 1. Последовательность слов, букв, цифр или других знаков, написанных в одну линию. Например, битовая строка — последовательность двоичных цифр (нулей и единиц). 2. Тип данных, значениями которого являются последовательности знаков. Обычно реализуется в виде одномерного массива переменной длины, минимальный размер которого равен единице, а максимальный размер (длина строки), может изменяться. 3. Горизонтальная линия на экране дисплея или на бумаге, заполненная последовательностью знаков или предназначенная для такого заполнения. Например, командная С., С. подсказки, или С. текста при работе в текстовом редакторе либо напечатанная на принтере.

Строка меню — элемент управления в графическом интерфейсе пользователя, реализующий функции горизонтального меню.

Строка подсказки (в интерактивных системах) — строка на экране дисплея, указывающая доступные команды и их смысл. Например, при наведении указателя мыши на одну из кнопок графического меню в С. п. появляется текст, поясняющий, какие действия вызовет «нажатие» этой кнопки.

Структура данных (Data structure) — организационная схема записи или массива, в соответствии с которой упорядочены данные, с тем чтобы их можно было интерпретировать и выполнять над ними определенные операции.

Структура базы данных — принцип или порядок организации записей в базе данных и связей между ними.

Структурное программирование — метод разработки программ, требующий разбиения программы на небольшие независимые части (модули). Структур-

ное программирование обеспечивает возможность проведения строгого доказательства правильности программ.

Сценарий — 1. План работы программы во взаимодействии с пользователем. Например, создание компьютерной игры начинается с разработки С., описывающего, что и в какой последовательности должна выполнять ЭВМ в зависимости от игровой ситуации и реакции играющего. 2. Данные, описывающие последовательность связанных событий.

Т

Таймер — программируемое устройство отсчета времени, часы. Т. обеспечивает измерение астрономического времени и позволяет предварительно установить временной интервал, по истечении которого может быть выдан соответствующий сигнал. Этот сигнал можно использовать в программах для прерывания или начала некоторых запрограммированных действий.

Текстовое поле — элемент управления, предназначенный для ввода, отображения и редактирования небольших текстовых значений.

Текстовый редактор — программа для ввода и изменения текстовых данных: документов, книг, программ и т. д. Редактор обеспечивает модификацию строк текста, контекстный поиск и замену частей текста, автоматическую нумерацию страниц, обработку и нумерацию сносок, выравнивание абзацев, проверку правописания слов, построение оглавлений, распечатку текста на принтере.

Текстовый режим — режим работы адаптера дисплея, при котором на экране выводятся изображения только текстовых и псевдографических символов.

Текстовый файл — 1. Файл, содержащий текст. 2. Файл, состоящий исключительно из кода ASCII.

Текущий дисковод — дисковод, с которым в данный момент непосредственно работает операционная система. В IBM PC это дисковод, с которым в настоящий момент работает пользователь.

Текущий каталог — каталог, файлы которого в данный момент непосредственно доступны программам и пользователю.

Тело процедуры — исполняемая часть процедуры, представляющая последовательность операторов, реализующую алгоритм процедуры.

Тело цикла — последовательность повторяемых в цикле команд или операторов.

Тест (Test) — совокупность входных данных для программы, а также точное описание всех результатов, которые должна выработать программа на этих данных.

Тестирование (Testing) — этап разработки программы, в процессе которого проверяется работоспособность программы, не содержащей явных ошибок.

Тип данных (Data type) — характеристика набора данных, которая определяет диапазон возможных значений данных из набора, допустимые операции, которые можно выполнять над этими значениями, способ хранения этих значений в памяти.

Тип файла — характеристика файла, отражающая его назначение и область применения, например командный, текстовый, Pascal-программа и т. п.

Точка возврата — место в вызывающей программе, в которое осуществляется возврат из подпрограммы после ее завершения.

Транслятор — программа, преобразующая текст, написанный на алгоритмическом языке, в программу, состоящую из машинных команд.

Трансляция программы — перевод программы с одного языка программирования на другой. Обычно Т. является преобразованием программы, написанной на машинно-независимом языке, в эквивалентную программу на машинном языке конкретной ЭВМ.

Трассировка программы — выполнение программы или ее участка, сопровождающееся выводом на экран, принтер или другой регистрацией в хронологической последовательности информации о событиях, связанных с выполнением программы. Т. применяется при отладке или тестировании программы, когда программа пользователя или ее отлаживаемый участок выполняется под управлением специальной программы-трассировщика.

У

Указатель мыши — значок на экране дисплея, передвигающийся при движении мыши по плоскости.

Управляющая клавиша — клавиша, которая, будучи нажатой в сочетании с другой клавишей, придает последней альтернативный смысл.

Условие выхода из цикла (условие завершения цикла) — условие, в зависимости от выполнения которого происходит или повторение выполнения тела цикла, или выход из цикла.

Ф

Файл (File) — совокупность связанных записей, хранящихся во внешней памяти компьютера и рассматриваемых как единое целое. Каждый файл однозначно идентифицируется указанием имени файла, его расширения и пути доступа к файлу.

Файл исходных данных — совокупность записей, расположенных в порядке их получения в пункте записи.

Файл последовательного доступа (последовательный файл) — файл, записи которого можно читать только последовательно, друг за другом, в порядке их расположения.

Файл прямого доступа — файл, записи которого можно читать в произвольном порядке. В таком файле каждая запись снабжена своим номером — ключом, по которому и осуществляется поиск нужной записи.

Файл расчетных данных — совокупность записей, элементы которой получены путем обработки исходных или инвертированных файлов.

Флажок — элемент управления, предназначенный для выбора одного из двух возможных значений.

Форма — окно, которое является основой проектируемого приложения Delphi. На форме размещаются компоненты — элементы создаваемого приложения.

Формат (Format) — структура информационного объекта. Формат определяет способ расположения и представления данных в таблицах, базах данных, принтерах, блоках данных и других информационных объектах.

Форматирование текста — процесс придания тексту определенного вида, связанный с определением левой и правой границ текста, абзацного отступа и т. д.

Ц

Центральный процессор (Central Processing Unit) — основной рабочий компонент компьютера, который выполняет арифметические и логические операции, заданные программой, управляет вычислительным процессом и координирует работу всех устройств компьютера.

Цикл — оператор языка программирования, позволяющий многократно повторять одну и ту же последовательность команд (тело цикла).

Цикл со счетчиком (арифметический цикл) — при выполнении такого цикла определенная часть программы периодически повторяется, а число повторений (проходов) регистрируется счетчиком. По достижении нужного количества проходов выполнение цикла прекращается. Текущее значение счетчика часто используется в теле цикла. Ц. с. с. могут быть как циклами с предусловием, так и циклами с постусловием.

Цикл с постусловием (цикл «до») — цикл, в котором проверка условия выхода из цикла осуществляется в конце выполнения тела цикла.

Цикл с предусловием (цикл «пока») — цикл, в котором проверка условия выхода из цикла осуществляется в начале выполнения тела цикла.

Ш

Шаблон — в информатике — формализованный кадр изображения, выводимый на экран дисплея и содержащий тексты запросов к пользователю и специальные поля, предназначенные для занесения туда ответов пользователя (текстов или чисел).

Шаблон имени файла — последовательность знаков, позволяющих обозначить имена сразу нескольких файлов. Ш. и. ф. образуется из символов, присутствующих в имени файла, и знаков, указывающих, что на их месте в имени символы либо отсутствуют, либо могут стоять любые допустимые символы.

Шрифт — конкретный способ изображения символов из некоторого набора. Обычно в Ш. входят буквы одного или нескольких алфавитов, цифры и специальные знаки.

Щ

Щелчок — действие, заключающееся в быстром нажатии и отпускании кнопки неподвижной мыши. Щ. применяется при задании команды с помощью элемента управления, быстрой активизации окон и в других случаях. При этом указатель мыши должен указывать на выбранный элемент или программу.

Э

Электронные таблицы (Spreadsheet) — компьютерная программа, поддерживающая представление данных в виде таблиц, состоящих из строк и граф, на пересечении которых располагаются клетки (ячейки таблицы). Значение в числовой клетке таблицы либо указывается в явном виде, либо рассчитывается по ассоциированной с клеткой формуле. Электронные таблицы являются инструментом анализа (финансовой) информации.

Элемент данных — неделимый информационный элемент, являющийся минимальной структурной единицей информации. Вид элемента данных определяется характером содержащихся в нем сведений и особенностью его организации или записи.

Элемент управления — средство графического интерфейса пользователя, предназначенное для ввода-вывода информации и управления работой программы. В настоящее время используются множество различных Э. у., например, кнопки, флажки, переключатели, поля, списки, полосы прокрутки.

Я

Язык ассемблера (Assembler) — алгоритмический язык низкого уровня, основной состав исполнительных операторов которого базируется на машинных командах.

Язык программирования (Programming language) — искусственный (формальный) язык, предназначенный для записи алгоритмов. Язык программирования задается своим описанием и реализуется в виде специальной программы: компилятора или интерпретатора.

Язык программирования высокого уровня — язык программирования, в который введены элементы, допускающие описание задачи в наглядном, легко воспринимаемом виде, упрощающие и автоматизирующие процесс программирования. Управляющие конструкции и структуры данных такого языка отражают естественные для человека понятия, а не архитектуру вычислительной системы.

Язык описания данных — язык для описания физической и/или логической структуры данных.

Приложение Б

Меню интегрированной системы программирования Turbo Pascal 7.0

Turbo Pascal дает возможность легко и эффективно писать, редактировать, компилировать, редактировать связи и отлаживать программы. Все это позволяет делать интегрированная среда программирования фирмы Borland, коротко IDE. Для ознакомления со средой запустите программу TPTOUR. Она продемонстрирует возможности интегрированной среды программирования и покажет, как открыть файлы, редактировать их, компилировать, запускать и отлаживать программы, к тому же сформирует у вас навыки общего управления окнами.

Для детального изучения пакета постоянно используйте встроенную систему справочной информации Turbo Pascal. Так как она является контекстно-ориентированной, то, выбрав нужный пункт меню интегрированной среды и нажав клавишу F1, вы получите справочную информацию о назначении данного пункта.

После запуска среды программирования Turbo Pascal 7.0 в верхней части экрана отображается меню:

File Edit Search Run Compile Debug Tools Options Window Help

Рис. Б.1. Меню среды программирования Turbo Pascal 7.0

Выбор команды меню выполняется любым из стандартных способов: F10, Alt+горячая клавиша или щелчком мышью по нужному пункту меню. Назначение команд меню представлено в табл. Б.1.

Таблица Б.1. Назначение команд меню среды программирования Turbo Pascal 7.0

Раздел меню	Назначение
Меню File	Меню File позволяет открывать и создавать файлы с программами в окнах редактора, а также сохранять изменения, выполнять другие файловые функции, осуществлять временный выход в DOS и выход из среды программирования
New	Открывает новое окно редактирования с именем по умолчанию NONAMEXX.PAS (вместо XX используются числа от 0 до 99) и автоматически делает его активным

Раздел меню Назначение

Open
Отображает диалоговое окно выбора файла, который будет открыт в окне редактора. Это диалоговое окно содержит окно ввода, список файлов, кнопки Open (Открыть), Replace (Заменить), Cancel (Отказ), Help (Справочная информация) и информационную панель, описывающую выбранный файл.

Можно выполнить любое из следующих действий:

- ввести полное имя файла и выбрать Replace (заменить) или Open (открыть). Open загружает файл в новое окно редактора. При выборе Replace окно редактора должно быть активным; содержимое окна заменяется выбранным файлом;
- ввести имя файла со спецификатором (знаком вопроса или звездочкой), что позволяет отфильтровать список файлов в соответствии с заданным шаблоном;
- выбрать стрелку вниз для выбора файла из списка файловых спецификаций, введенного ранее;
- просмотреть содержимое различных каталогов путем выбора имен каталогов из списка файлов.

Для выбора имени нужно дважды щелкнуть по нему мышью или применить клавиши со стрелками и нажать Enter.

Как только требуемый файл выбран, следует нажать кнопку Open (или Cancel, в том случае, если вы изменили свое решение)

Save
Сохраняет файл, находящийся в активном окне редактора, на диск. (Этот элемент меню недоступен, если нет активного окна редактора.) Если файл имеет имя по умолчанию (NONAME00.PAS или подобное), Turbo Pascal откроет диалоговое окно Save File As (Сохранить файл как) для того, чтобы позволить вам переименовать файл и сохранить его в текущем каталоге или на другом устройстве. Это диалоговое окно подобно окну, открывающемуся для команды Save As (Сохранить как), описанной ниже

Save As
Сохранить файл, находящийся в активном окне редактора, под другим именем, в другом каталоге, на другом устройстве. После выбора этой команды отображается диалоговое окно Save File As, в котором нужно ввести новое имя, не обязательно с устройством и каталогом, и нажать OK. Все окна, содержащие этот файл, обновятся с этим новым именем. Если выбрано имя существующего файла, файл будет записан на место существующего

Save all
Работает точно так же, как команда Save, за исключением того, что она сохраняет все модифицированные файлы, а не только файл, находящийся в активном окне редактора. Эта команда недоступна, если ни одно окно редактора не открыто

Change dir
С помощью диалогового окна Change Directory можно задать текущее устройство и каталог. Текущий каталог — это каталог, который Turbo Pascal использует для сохранения файлов и их поиска. (При использовании относительных путей в Options ► Directories они относятся только к текущему каталогу). При этом существует два способа смены каталогов: ввести путь доступа нового каталога в окне ввода и нажать Enter, или выбрать требуемый каталог в дереве каталогов (если вы используете клавиатуру, нажмите Enter, чтобы сделать каталог текущим), а затем выбрать OK или нажать Esc для выхода из диалогового окна. Если вы изменили свое решение и хотите вернуться к предыдущему каталогу (но вы еще не вышли из диалогового окна), нажмите кнопку Revert

Таблица Б.1 (продолжение)

Раздел меню	Назначение
Print	Печать содержимого активного окна редактора. Turbo Pascal расширяет табуляции (заменяет символы табуляции соответствующим количеством пробелов) и затем посылает их на печатающее устройство DOS. Эта команда недоступна, если активное окно нельзя распечатать. Используйте Ctrl+K+P для печати только выбранного текста
Printer setup	Открывает диалоговое окно установки принтера
DOS shell	Позволяет временно покинуть Turbo Pascal для выхода в DOS. Для возврата в Turbo Pascal наберите EXIT и нажмите Enter
Exit	Приводит к выходу из Turbo Pascal, удаляет его из памяти и возвращает управление в командную строку DOS. Если вы не сохранили сделанные изменения, то Turbo Pascal предлагает сохранить их перед выходом
Меню Edit	Выбирается нажатием Alt+E. Позволяет вырезать, копировать и вставлять текст в окне редактора. Можно также открыть окно Clipboard (буфер обмена) для просмотра и редактирования его содержимого. Перед использованием большинства команд этого меню необходимо выбрать текст (потому что большинство действий редактора применяются к выбранному тексту). Для выбора текста с помощью клавиатуры можно использовать следующие методы: ■ нажать клавишу Shift одновременно с клавишами со стрелками; ■ нажать Ctrl+K+B для отметки начала блока. Затем переместить курсор к концу блока и нажать Ctrl+K+K; ■ для выбора отдельного слова следует переместить курсор к слову и нажать Ctrl+K+T; ■ для выбора отдельной строки следует нажать Ctrl+K+L. Для выбора текста с помощью мыши: ■ перемещайте указатель мыши над требуемым текстом. Если необходимо продолжить выбор за пределами окна, перемещайте мышь в нужную сторону, и окно автоматически будет сдвигаться; ■ для выбора отдельной строки дважды щелкните по ней мышью; ■ для выбора нескольких строк текста нажмите кнопку мыши и перемещайте мышь по тексту; ■ чтобы расширить или сократить выбранный фрагмент, одновременно нажмите Shift и клавишу мыши в любом месте документа. Как только текст выбран, команды в меню Edit становятся доступными, а буфер обмена становится полезным. Буфер обмена — это специальное окно в Turbo Pascal, хранящее текст, который был удален или скопирован, так что можно вставить его в любом месте. Буфер обмена работает в тесном взаимодействии с командами редактора
Undo	Отменяет последнюю команду редактирования, выполненную над строкой (включая набор текста в пустой строке или Ctrl+Y). Эта команда работает только с последней модифицированной или удаленной строкой
Redo	Отменяет последнюю команду Undo в строке

Раздел меню	Назначение
Cut	Удаляет выбранный текст из документа и помещает этот текст в буфер обмена. Можно вставить этот текст в любой другой документ (или в другое место в этом же документе) посредством выбора команды Paste (Вставить). Текст остается в буфере обмена, так что можно вставлять один и тот же текст более одного раза
Copy	Оставляет выбранный текст нетронутым, но помещает его точную копию в буфер обмена. Можно вставить этот текст в любой другой документ посредством выбора команды Paste (Вставить). Можно также скопировать текст из окна справочной информации
Paste	Вставляет текст из буфера обмена в текущее окно в позицию, указанную курсором. Текст, который вставляется в настоящий момент, в окне буфера обмена помечен как текущий блок
Clear	Удаляет выбранный текст, не помещая его в буфер обмена. Это означает, что нельзя вставить этот текст, как в случае выбора команд Cut (Вырезать) или Copy (Копировать). Текст, удаленный с помощью этой команды, невозможновосстановим. Можно очистить сам буфер обмена посредством выбора всего текста в буфере обмена с последующим выбором команды Edit ► Clear
Show Clipboard	Открывает окно буфера обмена, который хранит текст, вырезанный и скопированный из других окон. Текущий выбранный текст — это текст, который Turbo Pascal использует при выборе команды Paste (Вставить). Можно отредактировать буфер обмена таким образом, чтобы вставляемый текст точно соответствовал требуемому. Окно буфера обмена похоже на любое другое окно редактора. Единственное отличие окна буфера обмена возникает при вырезании или копировании текста. Когда вы выбрали текст в окне буфера обмена и затем выбрали команду Cut (Вырезать) или Copy (Копировать), выбранный текст немедленно появляется в нижней части окна. (Помните, что любой текст, который был вырезан или скопирован, добавляется в конец буфера обмена, поэтому позднее его можно вставить.)
Меню Search	Выбирается нажатием Alt+S. Оно позволяет осуществлять поиск текста, объявления процедур и местоположения ошибок в файлах. После выбора этого пункта меню на экран выводится раскрывающееся меню
Find	Отображает диалоговое окно Find, позволяющее набрать текст, который требуется найти, и установить опции, влияющие на поиск. Диалоговое окно поиска содержит несколько кнопок: ■ Options (Опции); ■ Case sensitive — различать прописные и строчные буквы; ■ Whole words only — искать только целые слова; ■ Regular expression — регулярное выражение. Включайте эту кнопку в том случае, если хотите, чтобы Turbo Pascal распознавал спецификаторы ^, \$, ., *, +, [], \ в строке поиска. Они означают следующее: ^ — в начале строки — начало строки; \$ — в конце выражения — конец строки; . — любой символ.

Таблица Б.1 (продолжение)

Раздел меню	Назначение
	* — символ, сопровождаемый звездочкой, означает любое число повторений (включая ноль) этого символа. Например, bo* означает bot, b, boo, и даже be; + — символ, сопровождаемый знаком плюс, означает любое ненулевое число повторений этого символа. Например, bo+ означает bot или boo, но не be или b; [] — символы в квадратных скобках означают какой-то один символ, стоящий в этих скобках. Например, [bot] означает или b, или o, или t; [^] — этот знак в квадратных скобках в начале строки означает «не». Таким образом, [^bot] означает любой символ, кроме b, o или t; [-] — внутри квадратных скобок означает диапазон символов. Например, [b-o] означает любой символ от b до o; \ — этот знак перед символом спецификатора указывает Turbo Pascal на то, что нужно рассматривать этот символ как литеру, а не как спецификатор. Например, \^ означает ^, а не начало строки. Введите строку в окне ввода и выберите OK для начала поиска, а для отказа от поиска выберите Cancel. Если требуется ввести строку, поиск которой уже производился ранее, выберите стрелку вниз, чтобы посмотреть список и осуществить из него выбор. Можно также выбрать слово, на котором стоит курсор в окне редактора, и использовать его в окне поиска посредством вызова Find из меню Search. ■ Direction — выбор направления поиска (вперед, назад). ■ Scope — область поиска (весь файл или выделенный текст). ■ Origin — начало поиска (во всей области или от позиции курсора)
Replace	Отображает диалоговое окно, позволяющее вводить образец текста для поиска и замены. Диалоговое окно замены содержит несколько зависимых и независимых кнопок; многие из них идентичны кнопкам диалогового окна поиска, описанного ранее. Дополнительная кнопка Prompt on replace (Подсказка для замены) управляет подсказкой для каждой замены. Введите строку поиска и строку замены в окнах ввода и выберите OK или Change All (Заменить все) для начала поиска или нажмите Cancel для отказа. Если требуется ввести строку, использованную ранее, нажмите стрелку вниз для просмотра архивного списка. Если Turbo Pascal находит заданный текст, он спрашивает, хотите ли вы произвести замену. При выборе OK он находит и заменяет только первый образец элемента поиска. При выборе Change All (Заменить все) он заменяет все вхождения, как это определено с помощью кнопок Direction (Направление), Scope (Область) и Origin (Начало). Как и в диалоговом окне поиска, можно выбрать слово, на котором стоит курсор в окне редактора и использовать его в окне ввода Text to find (Текст для поиска) простым вызовом Find или Replace (Найти или заменить) из Search menu (Меню поиска). Можно добавить текст из окна редактора с помощью стрелки вправо
Search Again	Повторяет последнюю команду Find или Replace. Все установки, которые были сделаны в последнем диалоговом окне (Find или Replace), остаются в силе при выборе Search Again

Раздел меню	Назначение
Go to line number	Отображает подсказку номера строки, которую требуется найти
Show last compiler error	Отображает в верхней части экрана последнюю ошибку компилятора и позиционирует курсор возле ошибки. Если последняя компиляция была успешной, то сообщение не отображается
Find error	Находит местоположение ошибки этапа выполнения. Когда происходит ошибка этапа выполнения, адрес в памяти, где она произошла, отображается в формате seg:ofs (сегмент:смещение). При возврате в IDE Turbo Pascal автоматически определяет местонахождение ошибки. Эта команда позволяет найти ошибку снова, задавая значения seg и ofs. Чтобы Find error работала, нужно установить независимую кнопку Debugging в положение Integrated (в диалоговом окне Options ► Debugger). Если ошибки этапа выполнения произошли в программе, запущенной из IDE, то значения по умолчанию для адреса ошибки устанавливаются автоматически. Это позволяет вам переместить ошибку после изменения файлов. (Если вы просто переместились в том же самом файле, можете вернуться назад к местоположению ошибки с помощью команды Ctrl+Q+W.)
Find procedure	Отображает диалоговое окно, позволяющее ввести имя процедуры для поиска. Эта команда доступна только во время сеанса отладки. Введите имя процедуры или выберите стрелку вниз для выбора имени из архивного списка. В противоположность команде Find, эта команда находит объявление процедуры, а не пример ее использования
Меню Run	Выбирается нажатием Alt+R. Команды меню запуска позволяют запустить программу, а также начинают и заканчивают сеанс отладки. После выбора этого пункта меню на экран выводится раскрывающееся меню
Run	Запускает программу, используя параметры, переданные в нее с помощью команды Run ► Parameters. Если со времени последней компиляции исходный код был модифицирован, то встроенный менеджер проекта автоматически перекомпилирует и отредактирует программу. Если отлаживать программу не требуется, можно компилировать и редактировать ее с отключенными независимыми кнопками Debugging (что даст вашей программе больше памяти для выполнения) в диалоговом окне Option ► Debugger. Если вы компилируете программу с этой независимой кнопкой, установленной в Integrated, то результирующий исполнимый код будет содержать отладочную информацию, которая повлияет на поведение команды Run ► Run следующим образом: Если вы не модифицировали исходный код со времени последней компиляции, команда Run ► Run приведет к выполнению программы до следующей точки прерывания или к выполнению ее до конца, если точки прерывания не были установлены. Если со времени последней компиляции исходный код был модифицирован, а вы уже сделали несколько шагов по программе, используя команды Run ► Step Over или Run ► Trace into, то команда Run ► Run предложит перекомпилировать программу. Если вы ответите ДА, менеджер проекта перекомпилирует и отредактирует программу и установит ее выполнение с начала. Если вы ответите НЕТ, то программа будет выполняться до следующей точки прерывания или до конца, если точки прерывания не были установлены. Если вы не находитесь в активном сеансе отладки, то встроенный менеджер проекта перекомпилирует программу и установит ее запуск с начала

Таблица Б.1 (продолжение)

Раздел меню	Назначение
Program reset	Прекращает текущий сеанс отладки, освобождает память, отведенную программе, и закрывает все открытые файлы, используемые программой
Go to cursor	Выполняет программу до строки, на которой стоит курсор в текущем окне редактора. Если курсор стоит на строке, которая не содержит оператора, команда выдаст предупреждение. Команда Run ► Go to cursor может также начать сеанс отладки. Команда Go to cursor не устанавливает постоянную точку прерывания, но позволяет программе останавливаться на постоянной точке прерывания, если она встретила эту точку прерывания перед строкой, на которой стоит курсор. Если это произошло, следует выбрать команду Goto cursor снова
Trace into	Выполняет программу в пошаговом режиме. Когда она достигает вызова процедуры, то производится пошаговое выполнение процедуры, вместо выполнения процедуры как одного шага (см. Run ► Step over). Если оператор не содержит вызова процедур, доступных отладчику, Trace into остановится на следующем операторе. Используйте команду Trace into для перемещения к процедуре, вызываемой процедурой, которую вы сейчас отлаживаете. Если оператор содержит вызов процедуры, доступной отладчику, Trace into останавливается в начале определения процедуры. Следующая команда Trace into или Step over выполняет оператор в определении процедуры
Step over	Выполняет следующие операторы в текущей процедуре. Трассировка процедур нижнего уровня не выполняется, даже если они доступны отладчику
Parameters	Позволяет передать выполняющейся программе параметры командной строки точно так же, как в командной строке DOS. Команды переназначения DOS будут игнорироваться. При выборе этой команды отображается диалоговое окно с одним окном ввода
Меню Compile	Выбирается нажатием Alt+C. Используется для того, чтобы сделать Compile, Make или Build программы в активном окне. После выбора этого пункта меню на экран выводится раскрывающееся меню
Compile	Компилирует активный файл редактора. При этом на экран выводится окно статуса, показывающее результаты компиляции. Когда компиляция завершается, нажмите любую клавишу, чтобы удалить это окно. Если происходит какая-нибудь ошибка или предупреждение, окно редактора, содержащее исходный код с ошибкой, становится активным, появляется сообщение об ошибке, а курсор устанавливается на местоположении первой ошибки
Make	Вызывает встроенный менеджер проекта для создания .EXE файла. При этом выполняется следующее. Если для Primary File было задано имя, то компилируется этот файл; в противном случае компилируется файл в активном окне редактора. Turbo Pascal проверяет все файлы, от которых зависит компилируемый файл. Если исходный файл для данного модуля был модифицирован со времени создания файла .TPU (объектный код), то модуль перекомпилируется. Если для данного модуля был изменен интерфейс, то все другие модули, которые зависят от него, перекомпилируются. Если модуль, отредактированный в файл .OBJ (внешние программы), и файл .OBJ новее, чем файл .TPU этого модуля, то модуль перекомпилируется.

Раздел меню	Назначение
	Если модуль включает файл Include, а файл Include новее, чем файл .TPU этого модуля, то этот модуль перекомпилируется.
	Если исходный файл модуля (файл .TPU) не может быть установлен, то этот модуль не компилируется, но используется
Build	Перекомпилирует все файлы независимо от их даты. Эта команда подобна команде Compile ► Make за исключением того, что она не имеет условий
Destination	Позволяет определить, будет ли исполняемый код храниться на диске (как файл .EXE) или он будет храниться в памяти (и, таким образом, теряться при выходе из Turbo Pascal). Заметим, что даже если для Destination установлено значение Memory (память), все модули, перекомпилированные во время Make и Build, имеют свои обновленные файлы .TPU на диске. Если для Destination установлено значение Disk (диск), то создается файл .EXE, а его имя извлекается из одного или двух имен по следующему правилу: имя Primary File или, если оно не задано, имя файла в активном окне редактора
Primary file	Выбирается для того, чтобы задать файл .PAS, который будет компилироваться при использовании команд Compile ► Make (F9) или Build (Alt+C+B). Можно применять эту опцию при работе над программой, которая включает несколько модулей и файлов Include. Не имеет значения, какой файл вы только что редактировали; Make или Build всегда оперируют с приоритетным файлом
Clear primary file	Отменяет выбор файла Primary, после чего команды Compile и Build будут использовать файл в активном окне редактирования
Information	Открывает окно, в котором выдается информация о последней откомпилированной программе, текущем состоянии памяти и окружения
Меню Debug	Выбирается нажатием Alt+D. Команды меню отладки управляют всеми свойствами интегрированного отладчика. После выбора этого пункта меню на экран выводится раскрывающееся меню
Breakpoints	Открывает диалоговое окно, позволяющее управлять использованием безусловных точек прерывания. Оно показывает все установленные точки прерывания, номера их строк и условия. Условие имеет архивный список, позволяющий выбрать условие точки прерывания, использованное ранее. Когда бы выполняющаяся программа ни встретила точку прерывания, она остановится на строке с точкой прерывания. Перед компиляцией исходного файла можно установить точку прерывания на любой строке, даже на пустой строке или комментарии. При компиляции и выполнении файла Turbo Pascal проверяет все установленные точки прерывания и дает возможность удалить, игнорировать или изменить неправильные точки прерывания. При отладке файла Turbo Pascal «знает», какие строки содержат исполняемые операторы, и выдает предупреждение в случае попытки установить неправильные точки прерывания. Можно удалить точки прерывания из программы посредством выбора кнопки Delete (удалить). Можно также просмотреть исходный текст, где есть установленные точки прерывания, посредством выбора кнопки View (просмотр). View перемещает курсор к выбранной точке прерывания в окне Edit (эта команда не выполняет ваш код)

Таблица Б.1 (продолжение)

Раздел меню	Назначение
Call stack	Открывает окно, в котором показана последовательность процедур, вызываемых исполняемой программой. В окне содержатся имена процедур и значения передаваемых им параметров. Процедура, выполняемая в текущий момент времени, находится в вершине стека. Имя выполняемой программы находится в конце стека. Для показа текущей строки любой процедуры в окне выделите имя процедуры и нажмите Enter. Окно остается на экране в течение всего сеанса работы, если вы не закроете его. Команда Register открывает окно, показывающее регистры процессора, используемые обычно при отладке модулей на ассемблере. Верхняя половина окна показывает содержимое регистров, а нижняя — содержимое восьми флагов
Watch	Открывает окно, в котором содержатся выражения и их изменяющиеся значения. Элементы окна добавляются или удаляются командой Add Watch
Output	Открывает окно, в котором отображается вывод на экран только в текстовом режиме
User Screen	Отображает экран пользователя с возможностью наблюдения и текста и графики
Evaluate ► Modify	Вычисляет переменную или выражение, показывает ее значение, и, если это возможно, позволяет его изменить. Команда открывает диалоговое окно, содержащее три поля: Expression (выражение), Result (результат) и New Value (новое значение). Если отладчик может вычислить выражение, он отображает его значение в поле Result. Если выражение относится к переменной или простому элементу данных, то можно подвести курсор к полю New Value и ввести новое значение выражения. Для закрытия диалогового окна нажмите Esc
Add watch	Вставляет выражение для просмотра в окно Watch. При выборе этой команды отладчик открывает диалоговое окно и выдает подсказку для ввода выражения просмотра. Выражением по умолчанию является слово, на котором стоит курсор в текущем окне редактора. Имеется также архивный список, который можно применить для быстрого ввода выражения, использованного ранее. Если окно Watch является активным, можно вставить новое выражение для просмотра посредством нажатия клавиши Insert
Add breakpoint	Открывает диалоговое окно, в котором задаются параметры новой точки прерывания. В поле Condition вводится условие, при выполнении которого происходит прерывание. В поле Pass Count устанавливается число проходов контрольной точки, после выполнения которых произойдет останов. В поле File Name записывается полное имя исходного файла, содержащего текущую контрольную точку. В поле Line Number отображается номер строки, содержащей текущую точку прерывания. Можно ввести новое значение номера. Кнопка Modify предназначена для записи изменений, сделанных с помощью диалогового окна. Кнопка New служит для ввода информации для новой точки (условия, счетчик проходов, имя файла, номер строки)
Меню Tools	Выбирается нажатием Alt+T. Данное меню содержит различные отладочные команды

Раздел меню	Назначение
Messages	Открывает окно, в котором отображается информация из программы, выдаваемая с использованием фильтра DOS (типа GREP). Для отслеживания строк программы необходимо выбрать нужное сообщение и нажать клавишу Space. Чтобы отредактировать строку, на которую ссылается сообщение, укажите курсором нужное сообщение и нажмите Enter или дважды щелкните мышью
Go to next	Позволяет перейти к следующему элементу списка. Список содержит имена программ, которые можно запускать, не выходя из Turbo Pascal. Такие программы называются трансферными. После завершения такой программы выполняется возврат в среду программирования
Go to previous	Осуществляет переход к предыдущему элементу списка
Grep	Запускает поисковую утилиту, которая может осуществлять просмотр текста в нескольких файлах одновременно. Например, если вы забыли, в какой программе определена процедура SetUpModem, то можете использовать программу GREP для просмотра содержимого всех .PAS-файлов в каталоге на наличие строки SetUpMyModem. Эта программа установлена в меню Option ► Tools
Меню Options	Выбирается нажатием Alt+O. Это меню содержит команды, позволяющие посмотреть и изменить различные установки по умолчанию в Turbo Pascal. Большинство команд меню приводит к появлению диалогового окна
Compiler...	Отображает меню, которое предоставляет несколько опций, влияющих на компиляцию кода. Назначение этих опций следующее: Code Generation (генерация кода). Установки в этом окне определяют для компилятора методы создания объектного кода. Force Far Calls позволяет всем процедурам и функциям использовать дальние модели вызова. Если эта опция отключена, то компилятор использует ближние модели вызова для всех процедур и функций внутри компилируемого файла. (Эта опция эквивалентна директиве компилятора \$F.) Overlays Allowed делает возможной или невозможной генерацию оверлейного кода. Turbo Pascal разрешает делать модуль оверлейным только в том случае, если он был откомпилирован с включенной опцией Overlays Allowed. В этом состоянии генератор кода принимает особые предосторожности при передаче строк и установленных константных параметров из одной оверлейной процедуры или функции в другую. Включение независимой кнопки Overlays Allowed дает инструкции Turbo Pascal сделать модуль оверлейным, если это требуется. Если вы работаете над модулем, который планируете использовать и как оверлейный, и как неоверлейный, то переключение Overlays Allowed обеспечит вам и то, и другое при использовании одного и того же модуля. (Эта опция эквивалентна директиве компилятора \$O+). Word Align Data (когда опция включена) устанавливает выравнивание несимвольных данных по четным адресам. Когда эта опция отключена, Turbo Pascal использует побайтное выравнивание, где данные могут выравниваться по четным или нечетным адресам, в зависимости от того, какой следующий адрес является доступным. (Эта опция эквивалентна директиве компилятора \$A.)

продолжение ►

Таблица Б.1 (продолжение)

Раздел меню	Назначение
	Группа кнопок Run-time Errors позволяет выбрать, какие ошибки этапа выполнения будут генерироваться. Когда включена Range Checking, компилятор генерирует код, который проверяет, чтобы массив и строка не выходили за границы, а переменные скалярного типа не выходили за рамки заданных диапазонов. Если проверка была неудачна, программа прерывается с ошибкой этапа выполнения. (Эта опция эквивалентна директиве компилятора \$R.)
	При включении опции Stack Checking компилятор генерирует код, который проверяет перед каждым вызовом процедуры или функции, имеется ли доступное пространство для локальных переменных в стеке. Если проверка была неудачна, программа прерывается с ошибкой этапа выполнения. (Эта опция эквивалентна директиве компилятора \$S.)
	При включении опции I/O Checking компилятор генерирует код, который проверяет наличие ошибок ввода-вывода после каждого вызова ввода-вывода. Если проверка была неудачна, то программа прерывается с ошибкой этапа выполнения. Когда эта опция отключена, проверка ввода-вывода невозможна. Однако пользователь может тестировать ошибки ввода-вывода через системную функцию IOResult. (Эта опция эквивалентна директиве компилятора \$I.)
	Группа Syntax Options (синтаксические опции) позволяет выбрать тип синтаксических опций, по которым будет осуществляться поиск.
	При включенной опции Strict Var-String компилятор сравнивает объявленный тип строкового параметра типа var с типом действительно передаваемого параметра. Если они не идентичны, то происходит ошибка компиляции. Если эта опция отключена, то такая проверка не производится. (Эта опция эквивалентна директиве компилятора \$V.)
	С включенной опцией Complete Boolean Evaluation всегда вычисляются все термы в булевских выражениях. Если эта опция отключена, компилятор генерирует код, прекращающий вычисление булевского выражения, как только это становится возможным. (Эта опция эквивалентна директиве компилятора \$B.)
	При включенной опции Extended Syntax синтаксис Turbo Pascal расширяется таким образом, что позволяет использовать вызовы функций, определенных пользователем как операторы, как если бы они были процедурами. Если эта опция отключена, расширенный синтаксис невозможен. (Эта опция эквивалентна директиве компилятора \$X.)
	Опции Numeric Processing позволяют решить, как Turbo Pascal будет обрабатывать числа с плавающей точкой.
	Опции в группе Debugging (отладка) позволяют включить или отключить отладочную информацию или генерацию локальных символов.
	Включение Debug Information позволяет генерировать отладочную информацию, которая состоит из построчной таблицы для каждого оператора Pascal, отображающей адреса объектного кода в исходные текстовые числа. (Эта опция эквивалентна директиве компилятора \$D.)
	Когда вы включили Debug Information для данной программы или модуля, IDE позволит устанавливать точки прерывания в этом модуле.
	Когда происходит ошибка этапа выполнения в программе или модуле, откомпилированном с включенной опцией Debug Information, Turbo Pascal автоматически установит курсор на оператор, вызвавший ошибку, с помощью Search ► Find error.

Раздел меню	Назначение
	Включенная опция Local Symbols делает возможной генерацию информации о локальных символах, которая состоит из имен и типов всех локальных переменных и констант в модуле. (Эта опция эквивалентна директиве компилятора \$L.) При включенной опции Local Symbols IDE позволяет проверять и изменять локальные переменные модуля. Также вызовы процедур и функций модуля можно исследовать с помощью команды Window ► Call Stack. Информация о локальных символах записывается в файл .TPU вместе с объектным кодом модуля. Информация о локальных символах увеличивает размер файла .TPU и занимает дополнительную память при компиляции программы, использующей этот модуль, но не влияет на размер или скорость выполнения программы.
	Окно ввода Conditional Defines (условные определения) используется для того, чтобы ввести символы, на которые будут ссылаться директивы условной компиляции. При этом можно разделять условия точкой с запятой (;), например: TestCode; DebugCode
Memory sizes	Позволяет определить потребности памяти по умолчанию для программы. Все три установки можно задать в исходном коде, используя директиву компилятора \$M. Если программе не хватает динамической памяти, программа завершается с ошибкой этапа выполнения. (Эта опция эквивалентна директиве компилятора \$M.)
	В поле Stack Size задается размер (в байтах) сегмента стека. Размер по умолчанию 16,384, максимальный размер — 65,520.
	В окне Low Heap Limit задается минимальный требуемый размер динамической памяти (в байтах). По умолчанию минимальный размер равен 0 Кбайт.
	В окне High Heap Limit задается максимальный требуемый размер динамической памяти (в байтах). По умолчанию максимальный размер равен 655,360. Это значение должно быть больше или равно наименьшему размеру динамической памяти
Linker	Позволяет сделать несколько установок, влияющих на редактирование. Команда Linker открывает диалоговое окно, которое имеет несколько кнопок
Map File	Используется для выбора типа создаваемого файла .MAP. Для установок, отличных от Off, файл .MAP помещается в каталог EXE и TPU, заданный в диалоговом окне Options ► Directories. Установкой по умолчанию для файла .MAP является Off.
	Опция Link Buffer (буфер редактора связей) приводит к тому, что Turbo Pascal использует Memory (Память) или Disk (Диск) для буфера редактора связей. При выборе кнопки Memory увеличивается скорость компиляции, но может не хватить памяти для больших программ. Выбор кнопки Disk освобождает память, но замедляет компиляцию
Debugger	Открывает диалоговое окно, чтобы сделать несколько установок, влияющих на интегрированный отладчик.
	Кнопки Debugging определяют, будет ли отладочная информация включена в исполняемый файл и как будет выполняться .EXE-файл под управлением Turbo Pascal:
	■ выбор Integrated (по умолчанию) позволяет отлаживать программы как в интегрированном отладчике, так и в автономном отладчике Turbo Debugger;
	■ выбор Standalone дает возможность отлаживать программы только в Turbo Debugger.

Таблица Б.1 (продолжение)

Раздел меню	Назначение
	Онции Integrated, Standalone (Options ► Debugger) и Map File (Options ► Linker) создают полную информацию и информацию о локальных символах для модуля только в том случае, если этот модуль был откомпилирован с включенными опциями Debug Information и Local Symbols соответственно. Кнопки Display Swapping позволяют установить, когда интегрированному отладчику изменять окна просмотра во время выполнения программы. При отладке в режиме двойного монитора (с использованием опции /d в командной строке) вывод программы можно видеть на одном мониторе, а экран Turbo Pascal — на другом. В этом случае Turbo Pascal никогда не переключает экран, и установка Display Swapping ни на что не влияет. Если установить Display Swapping в None, то отладчик не будет переключать экран вообще. Нужно использовать эту установку для части отладки кода, где совершенно определено нет вывода на экран. При выполнении программы в режиме отладки с установкой по умолчанию отладчик Smart ищет в коде, который выполняется, места, где будет генерироваться вывод на экран. Если производится вывод на экран (или вызов процедуры), то происходит переключение с экрана интегрированной усовершенствованной среды на экран пользователя на время, достаточное для вывода, а затем экран переключается обратно. В противном случае переключения экрана не происходит
Directories	Определяет в Turbo Pascal, где искать файлы, необходимые для компиляции, редактирования связей и файлы вывода. Команда открывает диалоговое окно, содержащее четыре окна ввода. Используйте следующие директивы при введении имен каталогов в этих окнах ввода: ■ имена путей каталогов следует разделять точкой с запятой (;). Можно использовать максимум 127 символов (включая пробелы); ■ пробелы перед точкой с запятой и после разрешаются, но не требуются; ■ разрешаются относительные и абсолютные имена путей, включая имена путей, относящихся к фиксированной позиции на устройстве, отличном от текущего. Например: C:\PASCAL;C:\PASCAL\MYPROGS;A:\TURBO\EXAMPLES; Назначение каждого окна ввода: EXE and TPU Directory задает каталог вывода для файлов .EXE или .TPU. Если ввода в этом окне не было, файлы будут храниться в каталоге, где находятся исходные файлы. Файлы .MAP также будут храниться здесь, если Map File (Options ► Linker) установлена в значение, отличное от Off. Include Directories задает каталог, содержащий стандартные включаемые файлы. Стандартные включаемые файлы — это файлы, заданные директивой компилятора {\$I filename}. Разрешается несколько имен каталогов разделять точкой с запятой, как в команде DOS PATH. Unit Directories задает каталоги, содержащие ваши файлы модулей Turbo Pascal. При этом имена каталогов разделяются точкой с запятой (;), как в команде DOS PATH. Object Directory используется для задания каталогов, содержащих файлы .OBJ (подпрограммы ассемблерного языка). Когда Turbo Pascal встречает директиву {\$L filename}, он ищет сначала в текущем каталоге, затем в каталогах, заданных здесь

Раздел меню	Назначение
Tools	Открывает диалоговое окно, с помощью которого можно добавлять или убирать программы из меню Tools. Поле Program titles содержит список для обзора, добавления или удаления трансферных программ. Клавиша Edit позволяет изменить программу в списке и меню Tools. Клавиша New позволяет добавить новую программу. Клавиша Delete удаляет программу из списка и из меню
Environment	Дает возможность сделать установки для среды. Эта команда открывает меню, позволяющее выбрать установки для опций Preferences, Editor и Mouse. С помощью кнопок Screen Size можно установить, будет ли экран интегрированной усовершенствованной среды иметь 25 строк или 43/50 строк. В зависимости от типа видеоадаптера вашего PC будут доступны одна или две кнопки. При установке 25 строк (по умолчанию) Turbo Pascal использует 25 строк и 80 столбцов. При пошаговом прохождении или установлении местоположения ошибки в исходном коде интегрированная среда открывает новое окно, как только она встретит файл, который не был еще загружен. Выбор Current Window приводит к тому, что интегрированная среда заменяет содержимое самого верхнего окна редактора новым файлом вместо открытия нового окна редактора. Если Editor Files включена в опции Auto Save, и если файл был изменен со времени последнего сохранения, Turbo Pascal автоматически сохраняет исходный файл, как только выбрана команда Run ► Run (или любая другая команда отладки/выполнения) или File ► DOS shell. Когда включена опция Environment, все установки, сделанные во время этого сеанса работы, автоматически сохранятся в файле конфигурации TURBO.TP при выходе из Turbo Pascal. Когда включена опция Desktop, Turbo Pascal определяет, будет ли конфигурация сохраняться при выходе (в файле TURBO.DSK) и будет ли она восстанавливаться при возврате в Turbo Pascal. Ваша информация о конфигурации не будет сохраняться, если файл .TP не создан (автоматически или с помощью выбора команды Options ► Save Options), а кнопка Desktop Files не установлена в значение, отличное от None. Если вы хотите, чтобы IDE сохранила файл конфигурации окон и восстанавливала конфигурацию окон последнего сеанса программирования, выберите кнопку Current Directory или Config File Directory. Когда интегрированная среда сохраняет файл конфигурации TURBO.TP, она также создает TURBO.DSK, содержащий информацию об окнах редактора, позицию всех окон, архивные списки, положение точек прерывания и другую информацию. Все это можно сохранить и восстановить автоматически включением обеих опций Environment и Desktop в группе Auto Save. Альтернативно можно создать TURBO.TP, используя диалоговое окно Options ► Save Options. Когда в следующий раз вы загрузите TURBO.EXE, он будет искать TURBO.TP и TURBO.DSK в текущем каталоге. Если они будут найдены, то они загрузятся, а конфигурация предыдущего сеанса работы и конфигурация окон восстановятся. Если TURBO.TP не будет найден в текущем каталоге, IDE будет искать его в каталоге, содержащем сам TURBO.EXE.

Таблица Б.1 (продолжение)

Раздел меню	Назначение
	При выборе Editor в меню Environment появляются опции, из которых можно осуществить выбор. Диалоговое окно Editor Options имеет несколько кнопок, управляющих обработкой текста в окнах редактора.
	При включенной кнопке Create Backup Files (по умолчанию) Turbo Pascal автоматически создает копию исходного файла в окне редактора, если выбрана опция File ► Save, и дает этой копии файла расширение .BAK.
	При выключенной опции Insert Mode любой текст, набираемый в окнах редактора, будет перекрывать существующий текст. Когда эта опция включена, набираемый текст вставляется со сдвигом вправо. Нажатие Ins включает Insert-режим при работе в окне редактора.
	Когда включена опция Autoindent Mode, нажатие Enter в окне редактора устанавливает курсор под первый непустой символ в первой непустой строке. Это поможет сделать код вашей программы более читаемым.
	Когда включена опция Use Tab Character, Turbo Pascal вставляет символ табуляции (ASCII 9) при нажатии клавиши Tab. Если эта опция отключена, Turbo Pascal заменяет табуляцию пробелами, число которых определено установкой Tab Size, описанной ниже. Чтобы поменять способ, которым табуляция изображается в файле, просто измените значение размера табуляции на требуемый размер. Turbo Pascal пересчитает все табуляции в этом файле с учетом того размера, который был выбран. Можно сохранить этот новый размер табуляции в файле конфигурации посредством выбора команды Options ► Save Options.
	При включении опции Optimal Fill Turbo Pascal начинает каждую строку абзаца с минимально возможным количеством символов, используя табуляцию и пробелы по мере необходимости. Это приводит к созданию строк с меньшим количеством символов, чем при отключенной опции Optimal Fill.
	Когда включена опция Backspace Unindents (по умолчанию), а курсор стоит на непустой строке или первом непустом символе строки, нажатие пробела выравнивает строку до уровня без отступов.
	При включенной опции Cursor Through Tabs клавиши со стрелками передвинут курсор к середине табуляции, в противном случае курсор перепрыгнет несколько столбцов, пока не установится под табуляцией.
	При включении опции Use Tab Character в этом диалоговом окне и нажатии Tab Turbo Pascal вставляет символ табуляции в файл и передвигает курсор к следующему знаку табуляции. Окно ввода Tab Size позволяет установить, сколько символов до следующего знака табуляции будет пропущено. Допустимыми являются значения от 2 до 16; по умолчанию задано 8.
	При выборе Mouse из меню Environment отображается диалоговое окно Mouse Options, содержащее все установки для мыши.
	Кнопки Right Mouse Button определяют эффект нажатия правой клавиши мыши (или левой клавиши, если включена опция Reverse Mouse Buttons). По умолчанию задается Topic Search.
	В окне Mouse Double Click можно изменить положение бегунка на полосе управления для настройки скорости двойного щелчка мышью, используя клавиши со стрелками.

раздел меню	Назначение
	При включении кнопки Reverse Mouse Buttons активной кнопкой мыши становится самая правая вместо самой левой (установка для левши). Заметим, однако, что кнопки на самом деле не переключаются до тех пор, пока не будет выбрана кнопка OK.
	Выбор Startup из меню Environment позволяет выбрать установки для интегрированной среды. Все эти опции соответствуют вышеупомянутым опциям командной строки. Изменения, которые вы делаете здесь, записываются прямо в TURBO.EXE и не действуют до тех пор, пока вы не загрузите IDE снова.
	Диалоговое окно Colors используется для настройки цветов IDE. Список Group содержит имена различных областей IDE, которые можно настроить. При выборе этой группы окно списка Item будет содержать имена различных видов этой области. Можно менять основной и фоновый цвета, используя мышь или курсор для изменения палитры. Во всех системах текст в нижнем правом углу диалогового окна отражает текущие установки. Изменение не действует на конфигурацию окон до тех пор, пока вы не закроете диалоговое окно (путем выбора OK)
Open	Открывает диалоговое окно, с помощью которого можно отыскать установки, сохраненные ранее командой Save в файле с расширением .TP. В поле Option file name задается имя файла, в котором будут сохраняться установки. По умолчанию это TURBO.TP.
	Список Files содержит набор файлов текущего каталога, соответствующих маске, заданной в поле Option file name. На информационной панели отображается информация о полном имени, дате, времени и размерах выбранного файла
Save TURBO.TP	Сохраняет установки, сделанные в диалоговых окнах Find и Replace (из меню Search), Destination и Primary File (из меню Compile) и все установки в меню Options. В данном случае они хранятся в файле TURBO.TP
Save as	Открывает диалоговое окно, с помощью которого Turbo Pascal запоминает установки, сделанные в диалоговых окнах Find и Replace (из меню Search), Destination и Primary File (из меню Compile) и все установки в меню Options. В поле Option file name задается имя файла, в котором будут сохраняться установки. Список Files содержит набор файлов текущего каталога, соответствующих маске, заданной в поле Option file name
Меню Window	Выбирается нажатием Alt+W. Это меню содержит команды управления окном. Большинство из окон, которые вы откроете из этого меню, имеют все стандартные элементы окна: полосу прокрутки, закрывающую кнопку и кнопки масштабирования, позволяющие посмотреть и изменить различные установки по умолчанию в Turbo Pascal. Первые девять окон пронумерованы. Для выбора окна по номеру следует нажать Alt+N окна. Расположение открытых окон определяют команды Tile (неперекрывающееся расположение окон) и Cascade (расположение окон одно за другим с просмотром только активного окна, а для других окон видны только имена файлов и номера окон)
Close all	Закрывает все окна
Refresh display	Восстанавливает экран, если программа его случайно испортила
Size ► Move	Позволяет задать размер и позицию окна на экране

Таблица Б.1 (продолжение)

Раздел меню	Назначение
Zoom	Раскрывает активное окно во весь экран. Если окно уже расширено, то команда восстанавливает его текущий размер
Next	Активизирует следующее окно
Previous	Активизирует предыдущее окно, т. е. окно, бывшее активным перед текущим
Close	Закрывает текущее окно
List	Отображает список всех открытых окон
Меню Help	Выбирается нажатием Alt+H. Это меню дает доступ к встроенной справочной информации в специальном окне. Справочная информация имеется по всем аспектам интегрированной среды Turbo Pascal. (Также в строке статуса появляются однострочные подсказки для меню и диалоговых окон.)
Contents	Открывает окно Help с основной таблицей содержания. Из этого окна можно перейти к любой другой части системы справочной информации. Можно получить подсказку по справочной информации посредством нажатия F1, когда окно Help активно. Можно также получить этот экран, щелкнув мышью по строке статуса
Index	Открывает диалоговое окно, показывающее полный список ключевых слов справочной информации
Topic search	Показывает справочную информацию по языку и по текущему выбранному элементу. Чтобы получить справочную информацию по языку, установите курсор под элементом в окне редактора и выберите Topic Search
Previous topic	Открывает окно Help и вновь показывает текст, который вы просматривали последний раз. Turbo Pascal позволяет просмотреть 20 предыдущих экранов подсказки. Можно также щелкнуть мышью по строке статуса для просмотра последнего экрана справочной информации
Using help	Открывает текстовый экран, поясняющий, как пользоваться системой справочной информации. Если вы уже находитесь в системе справочной информации, можно вызвать этот экран нажатием F1
Files	Открывает диалоговое окно Install Help Files
Compiler directives	Открывает окно Help и выдает список директив компилятора, через которые можно выйти на соответствующий текст подсказки, относящийся к слову, помеченному курсором в списке
Reserved words	Открывает окно Help и выдает список слов, зарезервированных для внутреннего использования в языке. Пометив курсором соответствующее слово, можно вызвать поясняющий текст
Standard units	Открывает окно Help и выдает список стандартных модулей Turbo Pascal. Выделив соответствующий модуль, можно вызвать подсказку по размещенным в нем процедурам и функциям
Turbo Pascal Language	Открывает окно Help и выдает список элементов языка, через который можно выйти на соответствующий текст подсказки, относящийся к слову, помеченному курсором в списке
Error messages	Открывает окно Help и выдает информацию об ошибках, обнаруженных системой
About	Отображает диалоговое окно, которое покажет информацию о версии Turbo Pascal

Находясь в любом из окон среды программирования, можно получить дополнительный сервис через локальное меню, вход в которое реализуется нажатием Alt+F10 либо щелчком правой кнопкой мыши в окне или заголовке окна. Интегрированная среда программирования Turbo Pascal 7.0 предоставляет следующий набор локальных меню: Browse, Edit, Help, Message, Watch.

Таблица Б.2. Назначение команд локальных меню

Раздел меню	Назначение
Меню Browse	Открывается в окне просмотра и содержит следующие опции:
Browse	Позволяет посмотреть символ, отмеченный курсором
Previous	Выделяет предыдущий просмотренный объект
Goto source	Помещает курсор в выбранную строку исходного текста
Track source	Воздействует на ObjectBrowser, окно Message и отладчик. Если строка, на которую указано при просмотре, отсутствует в окне редактирования, то IDE открывает его и выделяет соответствующее место
Options	Открывает диалоговое окно, установки которого действуют только на то локальное окно просмотра, из которого открыто локальное меню
Меню Edit	Открывается в окне просмотра или строке меню и предоставляет набор команд для редактирования, оперативной подсказки и отладки
Меню Help	Открывается в окне Help и предоставляет услуги в виде опций меню, определяющих способ выдачи подсказки
Меню Message	Открывается в окне Message и выдает набор команд для управления точками наблюдения
Меню Watch	Открывается в окне Watch и предоставляет набор команд для редактирования выражений в окне просмотра, их блокирования и разблокирования

Рис. В.1. Меню ICP Delphi 6

Назначение каждого пункта меню можно уточнить в справочной системе Delphi. Для получения справки Delphi о пункте меню следует выбрать интересующий пункт меню, например, команду Run в меню Run, и нажать клавишу F1. После этого откроется окно Delphi Help, в котором будет выведена справочная информация.

Выбор команды меню выполняется любым из стандартных способов: F10, Alt+горячая клавиша или щелчком мыши по нужному пункту меню. Назначение команд меню представлено в табл. В.1.

Таблица В.1. Назначение команд меню ICP Delphi 6

Раздел меню	Назначение
Меню File	Разделы меню позволяют создать новый проект, новую форму, открыть ранее созданный проект или форму, сохранить проекты или формы в файлах с заданными именами
New	Создать новый объект из элементов депоzitария
Open	Открыть существующий модуль или проект
Open Project	Открыть существующий проект
Reopen	Открыть проект или модуль, с которыми работали раньше
Save	Сохранить текущий модуль
Save As	Сохранить текущий модуль под новым именем
Save Project As	Сохранить текущий проект под новым именем
Save All	Сохранить текущий проект и все его файлы
Close	Закрыть текущий модуль
Close All	Закрыть все файлы текущего проекта или группы
Use Unit	Сформировать оператор uses
Print	Напечатать текст модуля или форму
Exit	Завершить работу ICP Delphi
Меню Edit	Разделы этого меню позволяют выполнять обычные для приложений Windows операции с буфером обмена, а также дают возможность выравнивать группы размещенных на форме компонентов по размерам и местоположению
Undo	Отмена операции редактирования
Redo	Отказаться от отмены. Действует только немедленно после команды Отмена
Cut	Вырезать
Copy	Копировать
Paste	Вставить
Delete	Удалить
Select All	Выделить все

продолжение ➤

Приложение В

Справочные сведения по среде программирования Delphi 6

Для проектирования, запуска и тестирования приложений Delphi используется интегрированная среда разработки — ICP (Integrated Development Environment, IDE), в которой объединены редактор кода, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных — все, с чем приходится работать программисту при создании приложения. Такая интеграция предоставляет разработчику полный набор инструментов, дополняющих друг друга. Более того, в ICP имеется возможность расширять меню, включая в него вызов необходимых вам дополнительных программ, в том числе и ваших собственных.

Внешний вид среды программирования Delphi отличается от многих других сред программирования. Такие среды, как, например, Turbo Pascal 7.0, Word for Windows — являются MDI-приложениями и выглядят иначе, чем Delphi.

ПРИМЕЧАНИЕ

MDI (Multiple Document Interface) определяет особый способ управления нескольких дочерних окон внутри одного большого окна.

Среда Delphi является SDI-приложением (Single Document Interface) и состоит из нескольких отдельно расположенных окон. Это было сделано из-за того, что SDI близок к той модели приложений, что используется в Windows 95/98.

В главе 14 было рассмотрено назначение основных элементов ICP Delphi 6. В данном приложении мы рассмотрим назначение меню ICP Delphi, а также некоторых элементов, не описанных ранее.

Система меню

Строка главного меню ICP Delphi отображается в верхней части окна, как показано на рис. В.1.

Таблица В.1 (продолжение)

Раздел меню	Назначение
Align to Grid	Выровнять выделенные компоненты по сетке. Чтобы выбрать более одного компонента, при нажатой клавише Shift выделите мышью выравниваемые компоненты
Bring to Front	Переместить данный компонент на передний план (перед всеми другими компонентами на форме)
Send to Back	Переместить данный компонент на задний план (позади всех других компонентов на форме)
Align	Открыть диалоговое окно Выравнивание
Size	Открыть диалоговое окно Размер, чтобы задать одинаковые размеры нескольких компонентов
Scale	Открыть диалоговое окно Масштаб, чтобы пропорционально изменить размеры всех компонентов на текущей форме
Tab Order	Открыть диалоговое окно Edit Tab Order, чтобы изменить порядок перехода между компонентами на форме или внутри выбранного компонента, если этот компонент содержит другие компоненты
Creation Order	Открыть диалоговое окно Creation Order, чтобы определить порядок, в котором приложение будет создавать невидимые компоненты, когда вы загружаете форму во время разработки или во время выполнения
Flip Children	Зеркально повернуть компоненты в текущей форме «справа налево»
Lock Controls	Зафиксировать все компоненты на активной форме в их текущем положении
Add to Interface	Добавить новую процедуру, функцию или свойства для компонента ActiveX. Эти элементы будут добавлены в интерфейс компонента ActiveX, обеспечивая их доступ другим приложениям
Меню Search	Разделы этого меню позволяют осуществлять поиск фрагментов текста, ошибок, объектов, модулей, переменных и символов в редакторе кода
Find	Найти
Find in Files	Найти в файлах
Replace	Заменить
Search Again	Искать снова
Incremental Search	Быстрый поиск по вводимым символам
Go to Line Number	Перейти на строку с заданным номером
Find Error	Найти ошибку выполнения
Browse Symbol	Просмотр информации об идентификаторе
Меню View	Разделы этого меню позволяют вывести на экран или скрыть различные элементы среды проектирования Delphi и открыть окна, связанные с интегрированным отладчиком
Project Manager	Активизирует окно менеджера проекта (Project Manager)
Object Inspector	Активизирует инспектор объектов (Object Inspector)
Alignment Palette	Палитра выравнивания
Browser	Окно наследования

Раздел меню	Назначение
Code Explorer	Активизирует окно Исследователя кода, помогающего анализировать текст модуля
Component List	Список компонентов
Window List	Список открытых окон
Toggle Form/Unit	Переключение между формой и кодом ее модуля
Units	Список модулей проекта
Forms	Список форм проекта
Type Library	Библиотека типов
New Edit Window	Новое окно редактора кода
Toolbars	Делает видимыми кнопки оперативного доступа, если до этого они были невидимы
Меню Project	Разделы меню Project позволяют добавлять и удалять из проекта формы, задавать опции проекта, компилировать проект без его выполнения и выполнять другие полезные операции
Add To Project	Добавить в проект
Remove from Project	Удалить из проекта
Add To Repository	Добавить текущую форму в депозитарий
View Source	Занести в окно Редактора кода исходный файл проекта
Add New Project	Эта команда создает выбором из депозитария новый проект и добавляет его в текущую группу проектов
Compile	Компилировать все те файлы проекта, которые были изменены с момента последнего выполнения программы
Build All	Компилировать все компоненты, модули, формы вне зависимости от того, изменялось ли что-нибудь с момента последнего выполнения программы
Syntax Check	Проверить синтаксис приложения без генерации исполняемого файла
Information	Дать информацию о размерах приложения: числе строк кода, размере исполняемого модуля, размере стека и т. п.
Web Deploy	Развернуть спроектированную активную форму на веб-сервере
Options	Установить опции проекта (опции компиляции, редактирования связей и задать каталоги) в диалоговом окне опций проекта Project Options
Меню Run	Предоставляет возможность выполнять проект в нормальном или отладочном режимах, по шагам, останавливаясь в указанных точках, просматривая значения переменных и т. д.
Run	Выполнить приложение; если до этого не была осуществлена компиляция программы в ее текущем состоянии, то перед запуском эта компиляция выполняется
Parameters	Позволяет задать параметры командной строки, необходимые при запуске приложения
Register ActiveX Server	Зарегистрировать в Windows активный сервер ActiveX

Таблица В.1 (продолжение)

Раздел меню	Назначение
Un-Register	Снять с регистрации в Windows активный сервер ActiveX; тем самым экземпляр вашего активного элемента удаляется из системы
ActiveX Server	
Inspect	Открыть окно Инспектора отладки
Step Over	Выполнить приложение по шагам без трассировки функций
Trace Into	Выполнить приложение по шагам с трассировкой функций
Trace to Next Source Line	Выполнить приложение до следующей выполняемой команды
Run to Cursor	Выполнить приложение вплоть до той точки в исходном тексте, где находится курсор
Show Execution Point	Показать точку выполнения
Program Pause	Пауза в выполнении приложения
Program Reset	Завершить выполнение приложения и выгрузить его из памяти
Add Watch	Добавить переменную в окно наблюдения
Add Breakpoint	Добавить точку прерывания
Evaluate/Modify	Изменить значение переменной в процессе выполнения приложения
Меню Component	Содержит раскрывающееся меню, которое позволяет работать с компонентами: создавать новые компоненты, изменять палитру компонентов и т. п.
New Component	Создать новый компонент; активизируется Эксперт компонентов (Component Expert), который помогает создавать новые компоненты Delphi
Install Component	Установить новый компонент Delphi в новый или уже существующий пакет
Import ActiveX Library	Импортировать элемент ActiveX, который уже зарегистрирован в системе, в новый или уже существующий пакет Delphi в пакет
Create Component Template	Создать шаблон компонента; этот раздел становится активным, когда на форме выделено более одного компонента; он позволяет вам объединить их в один компонент, который может рассматриваться на форме как единое целое
Install Packages	Показать список всех имеющихся пакетов и указать, какие пакеты должны компилироваться в приложение; можно также просмотреть текущий список компонентов каждого пакета и редактировать состав пакетов
Configure Palette	Конфигурировать палитру компонентов, видимую на экране
Меню Database	В Delphi 6-3 позволяет использовать инструментарий для работы с базами данных. В Delphi 1 инструментарий для работы с базами данных включен в меню Tools (инструментарий)
Explore	Открыть SQL Explorer для создания, просмотра и редактирования данных
SQL Monitor	Контроль SQL запросов
Form Wizard	Мастер форм Базы данных позволяет легко генерировать форму, представляющую данные из любой базы данных
Меню Tools	Включает ряд разделов, позволяющих выполнять различные вспомогательные программы, например, вызывать Редактор изображений (Image Editor), работать с программами, конфигурирующими базы данных и сеть, и т. д. Кроме того, в это меню вы можете сами включить любые разделы, вызывающие те или иные приложения, и таким образом расширить возможности главного меню Delphi, настроив его для своих задач

Раздел меню	Назначение
Environment Options	Вызов диалогового окна настройки окружения Enviromental Options, позволяющего изменять установки редактора, экрана, палитры и опций просмотра
Repository	Вызов диалогового окна просмотра депоzitария Object Repository, позволяющего просмотреть помещенные в депозиторий объекты, добавлять и удалять объекты из депозитария
Configure Tool	Настройка раздела меню Tools, добавление в него новых разделов
Меню Window	Содержит список открытых окон ИСР Delphi и предоставляет возможность перехода из одного окна в другое
Меню Help	Содержит разделы, помогающие работать со справочной системой Delphi
Contents	Содержание справки
Keyword Search	Предметный указатель справки
What's New	Справка по новым возможностям Delphi
Getting Started	Начальное обучение
Using Object Pascal	Справка по Object Pascal
Developing Applications	Справка по разработке приложений
Object and Component Reference	Справка по объектам и компонентам библиотеки
Borland Home Page	Просмотр Web-страницы Borland
Delphi Home Page	Просмотр страницы Web Delphi
Borland Programs and Services	Просмотр страниц Web Programs и Services
About....	Справка о версии Delphi

ПРИМЕЧАНИЕ

Справочник Delphi является контекстно-зависимым: при нажатии клавиши F1 вы получите подсказку, соответствующую текущей ситуации. Например, находясь в Инспекторе объектов, выберите какое-нибудь свойство и нажмите F1 — вы получите справку о назначении данного свойства. Если в любой момент работы в среде Delphi возникает неясность или затруднение, можно нажать F1 и ознакомиться со справочной информацией.

Помимо главного меню в Delphi имеется система контекстных раскрывающихся меню, которые появляются, если пользователь поместил курсор мыши на том или ином компоненте и щелкнул правой кнопкой мыши. Большинство разделов этих контекстных меню дублируют основные разделы главного меню.

Палитра компонентов

Палитра компонентов — это своеобразная «витрина» библиотеки визуальных компонентов (Visual Component Library, VCL). Она использует постро-

начиную группировку объектов и позволяет сгруппировать компоненты в соответствии с их смыслом и назначением. Поскольку число предопределенных компонентов возрастает от версии к версии, то наиболее полной является библиотека Delphi 6, вид которой показан на рис. 14.3. Как показано на рис. В.2, в верхней части палитры находится набор закладок: Standard, Additional, Win32, System и т. д. Если щелкнуть мышью по одной из закладок, можно перейти на следующую страницу Палитры компонентов. Набор и порядок компонентов на каждой странице являются конфигурируемыми, то есть можно добавлять к имеющимся компонентам новые, изменить их количество и порядок.

Основная палитра компонентов Delphi имеет двенадцать страниц. Кратко рассмотрим содержание этих страниц.

Standard (Стандартная). Большинство компонентов на этой странице являются аналогами экранных элементов операционной системы Windows: меню, кнопки, полосы прокрутки, панели и т. п.

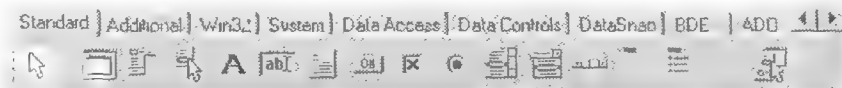


Рис. В.2. Палитра компонентов Standard

Имена компонентов, соответствующих значкам на палитре, можно узнать из всплывающей подсказки, появляющейся, если задержать над этим значком курсор мыши.

ПРИМЕЧАНИЕ

Имена на ярлычках выглядят, например, так: MainMenu, Button и т. д. Однако в действительности в Delphi названия всех имен классов начинаются с символа «Т», например, TMainMenu, TButton. Под такими именами во встроенной справочной системе Delphi вы сможете найти описания соответствующих компонентов.

Назначение компонентов можно уточнить, используя систему контекстной справки Delphi. Для этого следует выделить нужную страницу в палитре компонентов и нажать клавишу F1. После этого откроется окно Delphi Help со справочной информацией. Для просмотра информации в окне используйте полосу прокрутки. Если требуется просмотреть справку о конкретном компоненте, то следует выбрать нужную ссылку. Если выбрать компонент в палитре и нажать F1, то будет показана справка по типу данного компонента.

Страница Additional (Дополнительная) содержит более развитые компоненты (рис. В.3). Например, компонент Outline удобен для отображения информации с иерархической структурой, а компонент MediaPlayer позволит вашим программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых — отображение графической информации. Компонент Image загружает и отображает растровые изображе-

ния, а компонент Shape позволит добавить к вашим формам окружности, квадраты, и другие фигуры.



Рис. В.3. Палитра компонентов Additional

Палитра компонентов System (Системная) предоставляет возможность объединять отдельные элементы, такие как списки дисков, каталогов и файлов, для создания нестандартных диалоговых окон, связанных с обработкой файлов (рис. В.4). Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством технологии OLE (Object Linking and Embedding). А компонент Timer может генерировать события через определенные, заранее установленные промежутки времени.

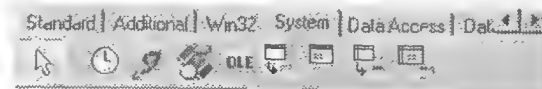


Рис. В.4. Палитра компонентов System

Страница Win32 (Windows 32) содержит компоненты, позволяющие созданному с помощью Delphi программам использовать такие нововведения в пользовательском интерфейсе 32-разрядной Windows, как просмотр древовидных структур, просмотр списков и панель состояния (рис. В.5).

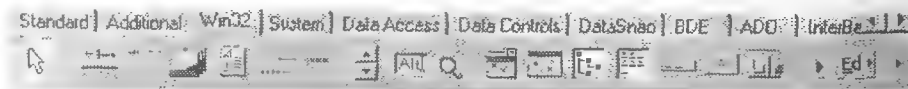


Рис. В.5. Палитра компонентов Win32

Страница Dialogs (Диалоговая) содержит компоненты — стандартные диалоговые окна для операций над файлами, поиска и замены текста, выбора шрифтов, цветов и т. д. (рис. В.6).



Рис. В.6. Палитра компонентов Dialogs

Data Access и Data Controls (Сервис баз данных). Delphi использует механизм баз данных компании Borland (Borland Database Engine, BDE) для организации доступа к файлам баз данных различных форматов (рис. В.7). Компоненты этих двух страниц облегчают программам Delphi использование сервиса

баз данных, предоставляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox.

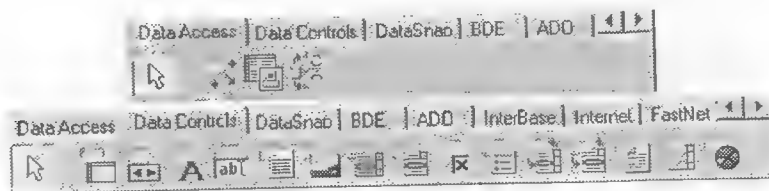


Рис. В.7. Палитры компонентов Data Access и Data Controls

Палитра **Data Access** содержит компоненты для работы с базами данных **Microsoft Access**. Палитра **Data Controls** содержит специализированные элементы системы управления базами данных, доступные вашим приложениям. С использованием этих компонентов создание программы просмотра и редактирования базы данных почти не требует программирования.

В Delphi 6 появился новый универсальный механизм доступа к данным — dbExpress.

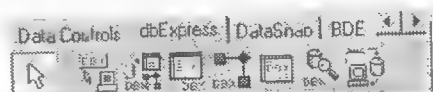


Рис. В.8. Палитра компонентов dbExpress

Для его применения в приложениях Delphi 6 предлагает компоненты доступа к данным и драйверы баз данных, реализующие интерфейсы этого универсального механизма доступа к данным с помощью клиентских API соответствующих серверных СУБД (рис. В.8). В комплект поставки Delphi 6 входят драйверы dbExpress для InterBase, Oracle, DB2, MySQL. Драйверы dbExpress представляют собой одну динамическую библиотеку, и обычно только она и требуется при поставке клиентских приложений, использующих dbExpress. Спецификация dbExpress разработана фирмой Borland, но является открытой, поэтому при необходимости можно создавать драйверы dbExpress для своих приложений.

Страница Win 3.1 содержит компоненты Delphi 1.0, возможности которых перекрываются аналогичными компонентами Windows 95 (рис. В.9).

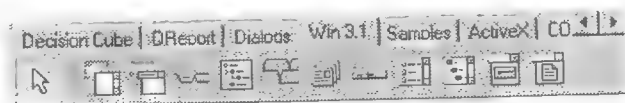


Рис. В.9. Палитра компонентов Win3.1

Страница Internet предоставляет компоненты для разработки приложений, позволяющих создавать HTML-файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Интернета (рис. В.10).



Рис. В.10. Палитра компонентов Internet

Компоненты этой страницы значительно облегчают создание обработчиков событий для обращения к определенному URL (Uniform Resource Locator, Унифицированный указатель ресурсов), представлению документов в HTML-формате и пересылки их клиентской программе.

Страница Samples (Примеры) содержит компоненты-примеры, которые вы можете добавлять в свои приложения (рис. В.10).



Рис. В.11. Палитра компонентов Samples

Исходный текст к этим типовым компонентам включен в каталог `.\SOURCE\SAMPLES` при установке по умолчанию.

Сраница ActiveX содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетка, диаграмма, средство проверки правописания (рис. В.11).



Рис. В.12. Палитра компонентов ActiveX

Страница QReport предоставляет компоненты для визуального проектирования отчетов баз данных. Здесь содержатся особые версии надписей, полей, примечаний и других элементов управления.

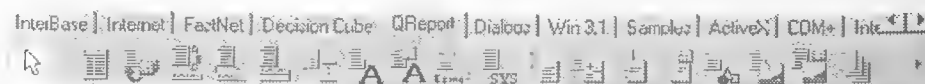


Рис. В.13. Палитра компонентов QReport

Страница Servers предоставляет компоненты-наследники ToleServer для доступа ко всем серверным объектам Microsoft Office.

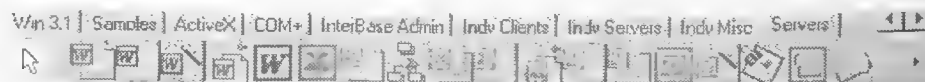


Рис. В.14. Палитра компонентов Servers

Дни компоненты автоматически запускают сервер при вызове одного из методов.

Новые компоненты

Библиотека компонентов Visual Component Library в Delphi 6 пополнилась новыми компонентами. Из компонентов, предназначенных для создания пользовательских интерфейсов Windows-приложений, следует особо отметить:

- **TLabeledEdit** — поле для ввода текста с меткой;
- **TValueListEditor** — компонент, предназначенный для отображения списка пар типа «параметр-значение», подобный представленному в Инспекторе объектов;
- **TComboBoxEx** — комбинированный список, позволяющий отображать рядом с текстом графические изображения;
- **TColorBox** — комбинированный список для выбора цвета.

Многие из компонентов приобрели дополнительные свойства (например, указывающие на то, как выглядят границы компонентов, такие как *BevelEdges*, *BevelInner*, *BevelOuter*, *BevelKind*, *BevelWidth*).

Еще одно новшество VCL — возможность использования в качестве свойств так называемых субкомпонентов, т. е. компонентов, владельцем которых является не форма, а другой компонент. Это означает, что теперь свойства субкомпонентов должны быть доступны в списке свойств компонентов-владельцев.

Появление в одной из предыдущих версий Delphi компонента *TActionList* существенно упростило создание пользовательских интерфейсов приложений. В Delphi 6 для работы с объектами *TAction* добавлены новые компоненты — *TActionManager* (для хранения коллекции объектов *TAction*); *TToolActionBar* и *TMainMenuActionBar* (интерфейсные элементы, предназначенные для предоставления пользователю доступа к функциональности, описанной в объектах *TAction*).

Окно редактора кода

Одной из наиболее важных элементов среды Delphi является окно Редактора кода. Окно Редактора кода в Delphi 6 показано на рис. В.15.

В действительности если вы откроете это окно в Delphi 6 в первый раз, оно может выглядеть несколько иначе и включать в себя еще одно встроенное окно — окно Исследователя кода (*Code Explorer*), которое отображает дерево всех типов, классов, реквизитов, методов, глобальных переменных и глобальных процедур, содержащихся в модуле, открытом в Редакторе кода. Исследователь кода упрощает исследование кода модуля и автоматизирует создание классов.

ПРИМЕЧАНИЕ

Во многих случаях вы можете закрыть это дополнительное окно, щелкнув мышью на кнопке в его правом верхнем углу, или задать опции среды проектирования, отменяющие появление окна *Code Explorer*.

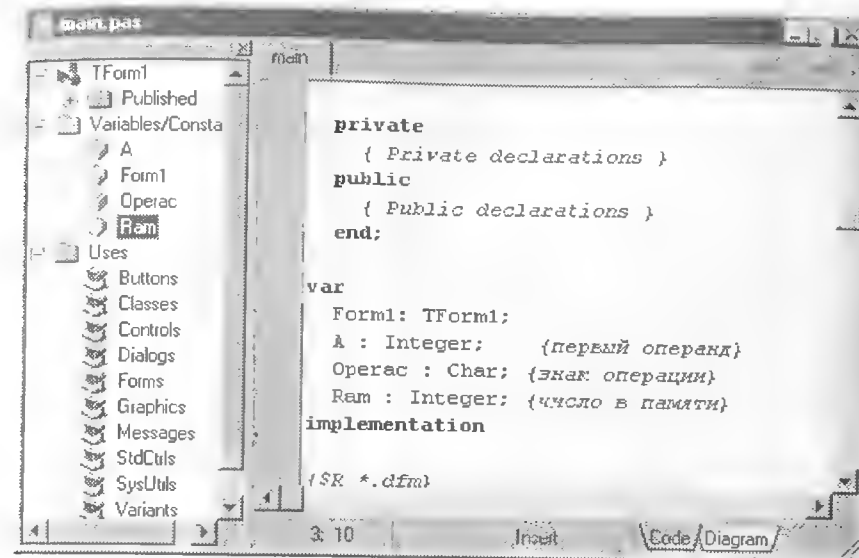


Рис. В.15. Окно Редактора кода с окном Исследователя кода (слева)

Редактор кода является полноценным программным редактором. Его можно настраивать на различный стиль работы, который вам более привычен. В редакторе применяется выделение цветом синтаксических элементов. Жирным шрифтом выделяются ключевые слова Object Pascal (на рис. В.15 вы видите выделение таких слов, как *private*, *public*, *end*, *var* и др.). Синим курсивом выделяются комментарии (например, на рис. В.15, это комментарии «{Private declarations}» или «{первый операнд}»).

В заголовке окна Редактора кода отображается имя текущего файла, с текстом которого производится работа. В верхней части окна можно видеть также закладки или ярлычки, указывающие текущую страницу (Main на рис. В.15). Приложения Delphi могут использовать много исходных файлов, и закладки помогают переходить от одного из них к другому. Можно также открыть дополнительное окно Редактора кода (командой *View ► New Edit Window* или щелчком в окне правой кнопкой мыши и выбором аналогичного раздела из контекстного меню) и одновременно работать с двумя модулями или с двумя разными фрагментами одного модуля.

В нижней части окна Редактора кода располагается строка состояния. В самой левой ее позиции отображается позиция курсора: номер строки и колонки. Второй элемент строки — индикатор модификации. Когда вы начинаете новый проект, то код, автоматически созданный Delphi, еще не сохранен. Вы должны сделать это сами командой *File ► Save*. Если код был изменен с того момента, когда вы в последний раз сохраняли его в файле, то в индикаторе модификации (правее индикатора строки и колонки) появляется слово

«Modified». Это означает, что код, который вы видите, изменен, и изменения не сохранены на диске. Третий элемент строки состояния — индикатор режима вставки. Это стандартный для большинства редакторов индикатор, который показывает, будут ли вводимые символы вставляться в текст (Insert) или заменять текст (Overwrite). Переключение режима Вставка/Замещение производится клавишей Insert.

В Delphi 6 Редактор кода позволяет выбрать режимы просмотра с помощью ярлычков в нижней части Редактора кода (список доступных режимов зависит от типа создаваемого приложения). В общем случае, помимо вывода собственно кода приложения, доступны следующие режимы отображения:

- **Diagram** — отображение связей между компонентами доступа к данным наподобие редактора модулей данных из предыдущей версии Delphi);
- **HTML Script** — отображение кода HTML и JavaScript, сгенерированного компонентами, которые предназначены для создания интерактивных веб-приложений;
- **HTML Result** — отображение HTML-кода, сгенерированного при использовании HTML-шаблона;
- **Preview** — отображение сгенерированного HTML-вывода так, как он выглядит в клиентском браузере.

В окно Редактора кода, как и в другие окна Delphi, встроена контекстная справка. Чтобы получить справку по какому-либо слову кода (ключевому слову Object Pascal, имени функции и т. п.), достаточно установить курсор на это слово и нажать клавишу F1.

Инспектор объектов

Инспектор объектов (Object Inspector) обеспечивает простой и удобный интерфейс для изменения свойств объектов Delphi и управления событиями, на которые реагирует объект. Как показано на рис. В.16, Инспектор объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница — это Properties (Свойства), вторая — Events (События). Для переключения между страницами свойств и событий используются закладки Properties и Events в верхней части Инспектора объектов. Выше страниц Properties и Events имеется раскрывающийся список всех компонентов, размещенных на форме. В нем можно выбрать тот компонент, свойства и события которого вас интересуют. Страница свойств (Properties) Инспектора объектов, показывает свойства того объекта, который в данный момент выделен.

Щелкните на окне формы и на странице свойств Инспектора объектов появятся свойства этой формы (они показаны на рис. В.16, а). Эти свойства можно изменять. Например, измените свойство Caption (надпись) вашей формы, на-

писав в ней текст «Калькулятор», и вы увидите, что эта надпись появится в полосе заголовка вашей формы.

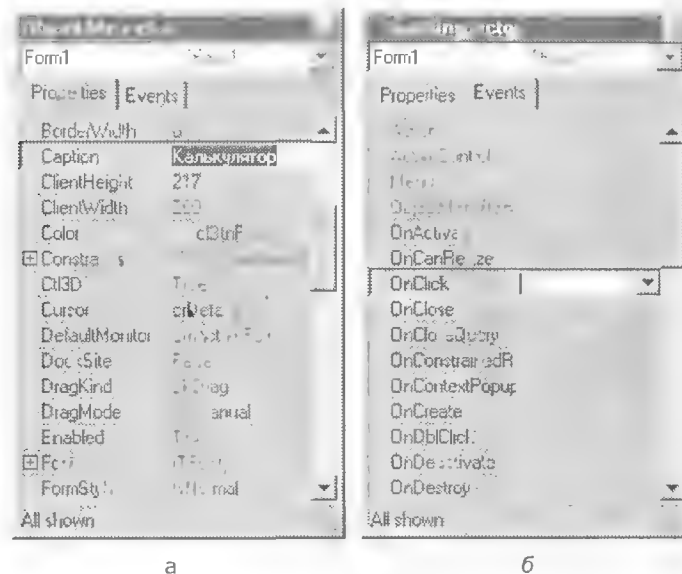


Рис. В.16. Окно Object Inspector: а — с раскрытой страницей Properties (Свойства); б — с раскрытой страницей Events (События)

Щелчок мышью по некоторым свойствам, например, по свойству Color, приводит к появлению справа от имени свойства раскрывающегося списка. Нажав в нем кнопку со стрелкой вниз, можно просмотреть список возможных значений свойства. Список значений можно также просмотреть с помощью бегунка. Например, измените значение свойства Color с принятого по умолчанию clBtnFace (цвет поверхности кнопок) на clWindow (цвет окна). Вы увидите, что поверхность формы изменит свой цвет.

Рядом с некоторыми свойствами можно видеть знак плюс (см., например, свойство Font на рис. В.16, а). Это означает, что данное свойство является объектом, который в свою очередь имеет ряд свойств. Щелкните мышью по этому плюсу или произведите двойной щелчок на свойстве Font. Будет открыта таблица таких свойств, как Color (Цвет), Height (Высота), Name (Имя шрифта), и др. Среди них можно видеть свойство Style (Стиль), около которого тоже имеется знак плюс. Щелчок мышью по этому плюсу или двойной щелчок по свойству открывает дополнительный список подсвойств, в котором можно, например, установить в true свойство fsBold (Жирный). Кстати, для смены true на false и обратно в подобных булевых свойствах не обязательно выбирать значение из раскрывающегося списка. Достаточно двойного щелчка мышью по значению свойства для его изменения. После того как вы

просмотрели или изменили подсвойства, можно опять произвести двойной щелчок мышью по головному свойству или щелкнуть мышью по знаку минуса около него, и список подсвойств свернется.

Страница событий (Events) представляет собой вторую часть Инспектора объектов (см. рис. В.16, б). На ней указаны все события, на которые может реагировать выбранный объект. Страница событий связана с Редактором кода следующим образом: если дважды щелкнуть мышью по правой стороне какого-либо пункта, то соответствующий данному событию код будет автоматически помещен в окно Редактора кода, а окно Редактора кода немедленно получит фокус, и вы сразу же будете иметь возможность отредактировать код обработчика данного события. Например, если требуется выполнить какие-либо действия при щелчке левой кнопкой мыши по данному объекту, то вы должны выделить событие `OnClick`. Рядом с именем этого события откроется окно с раскрывающимся списком. Если вы уже написали в своем приложении какие-то обработчики событий и хотите при событии `OnClick` использовать один из них, то можно выбрать необходимый обработчик из списка. Если же требуется написать новый обработчик, то необходимо произвести двойной щелчок мышью по пустому окну списка. После этого вы попадете в окно Редактора кода, в котором увидите текст:

```
procedure TForm1.FormClick(Sender: TObject);
begin
```

```
end;
```

Курсор будет расположен в пустой строке между ключевыми словами `begin` и `end`. Этот код является заготовкой обработчика события, автоматически сгенерированной Delphi. Вам остается только в промежутке между `begin` и `end` разместить необходимые операторы.

Теперь можно вернуться в Инспектор объектов, выделить в нем, например, событие `OnActivate` и нажать кнопку раскрывающегося списка. Вы увидите в нем введенный ранее обработчик события `FormClick`. Если требуется использовать тот же самый обработчик и в событии `OnActivate`, следует просто выбрать его из списка. Таким образом, можно избежать дублирования в программе одних и тех же фрагментов кода.

Пользуясь Инспектором объектов, можно получить контекстную справку по свойствам или событиям. Для этого выделите в окне Инспектора объектов интересующее вас свойство или событие и нажмите клавишу `F1`, а затем просмотрите в окне Delphi Help справочную информацию.

В Delphi 5-6 в Инспектор объектов введена возможность фильтрации свойств и событий и возможность группировать их по категориям. Для того чтобы воспользоваться этими возможностями, щелкните правой кнопкой мыши в окне Инспектора объектов. В появившемся меню можно выбрать раздел

`View`. Вам будет показан ряд категорий свойств и событий. Как показано на рис. В.17, около каждой категории имеется индикатор.

Можно включить индикаторы только у некоторых категорий, и тогда в Инспекторе объектов вы увидите события и свойства только указанных категорий. Выбор раздела `Toggle` позволяет переключать видимость разделов: те, которые были видимы, станут невидимы, и наоборот. Выбор раздела `All` делает видимыми все свойства и события, а выбор раздела `None` сделает все события и свойства невидимыми. В нижней части окна Инспектора объектов указывается, сколько свойств или событий невидимо в данный момент. В том же меню, появляющемся при щелчке правой кнопкой мыши в окне Редактора кода, можно выбрать раздел `Arrange` и в нем установить одну из двух возможностей: `by Name` — упорядочить свойства и события в алфавитной последовательности их имен, или `by Category` — упорядочить их по категориям. При упорядочивании по категориям форма представления событий и свойств кардинально меняется. При этом в окне отображаются категории с символами «+», при щелчке по которым раскрывается список элементов, относящихся к данной категории.



Рис. В.17. Изменение состава видимых свойств и событий

При этом некоторые свойства могут попасть одновременно в несколько категорий, но это не имеет значения: вы можете менять их значения в любой категории и они синхронно изменятся во всех остальных категориях.

Менеджер проектов

Менеджер проектов Delphi 6 позволяет работать одновременно над несколькими проектами, выбирая активный проект из раскрывающегося списка, как показано на рис. В.18.

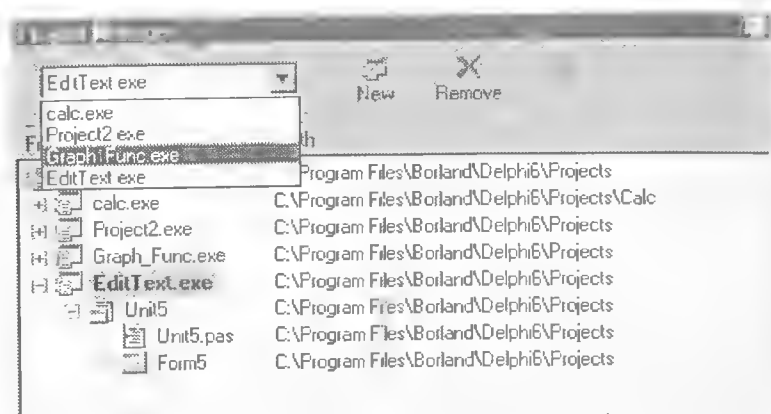


Рис. В.18. Выбор активного проекта в окне Менеджера проектов

Браузер проектов

В Delphi 6 для просмотра объектов используется Браузер проектов (Project Browser). В нем разработчику приложения предоставляется возможность выбора, какие именно объекты отображать — классы данного приложения или все классы VCL. С помощью двойного щелчка мышью по имени класса можно открыть отдельное окно с детальной информацией о нем.

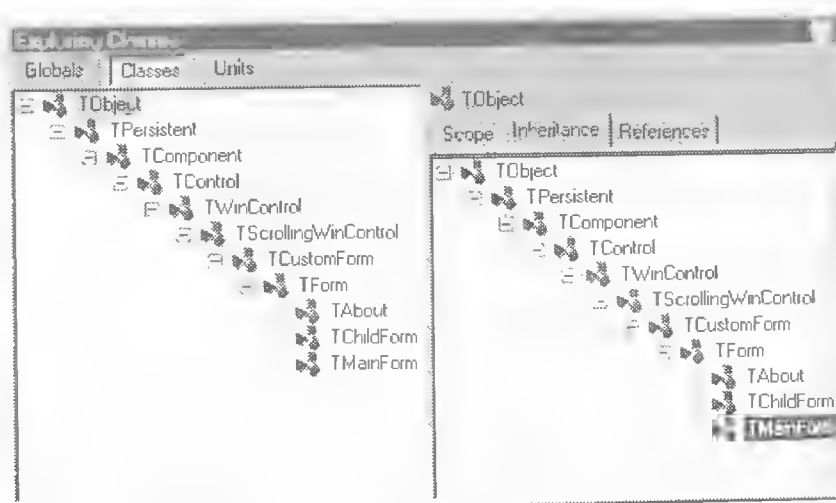


Рис. В.19. Окно Браузера проектов

Помимо просмотра иерархии классов, возможен просмотр глобальных переменных и модулей, а также получение сведений о том, какие другие модули ссылаются на данный модуль.

To-Do List: список недоделанных дел

Delphi 6 поддерживает так называемый список заданий (To-Do List). Задания могут относиться как ко всему проекту в целом, так и к отдельному модулю. Каждое задание может содержать имя исполнителя, приоритет, категорию задания (их может быть сколько угодно), текст задания и отметку, выполнено ли оно. Для добавления записи в список недоделанных дел можно выбрать в контекстном меню редактора кода команду Add To-do Item или нажать комбинацию клавиш Ctrl+Shift+T (рис. В.20).

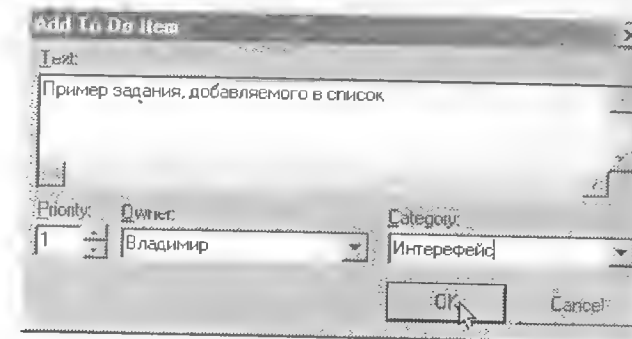


Рис. В.20. Создание нового задания

Список заданий можно просмотреть, выбрав пункт меню View ► To-do list. В этом окне можно добавлять элементы в список, отмечать их как выполненные, редактировать и менять их приоритетный порядок. Список заданий может быть отсортирован по параметрам: имя исполнителя, приоритет, категория задания. Если модуль, к которому относится задание, не загружен в редактор кода, то соответствующая строка в списке отображается серым цветом.

ПРИМЕЧАНИЕ

Список заданий можно скопировать в буфер обмена в виде текста или HTML-таблицы. Задания к отдельным модулям хранятся в качестве комментариев в исходном тексте модуля. Задания для всего проекта хранятся в файле с расширением .TODO. В списке задания для всего проекта и для отдельных форм сопровождаются разными значками.

Перетаскивание и встраивание окон

В Интегрированной Среде Разработки Delphi, начиная с Delphi 4, как и в оконных компонентах Delphi, широко используется технология Drag&Drop — перетаскивание и встраивание окон. Одно из встраиваемых окон — окно Исследователя кода, Code Explorer. По умолчанию оно встроено в окно Редактора кода. Если оно отсутствует на экране, то следует выполнить команду View ► Code Explorer.

Есть также еще много встраиваемых окон: окно Менеджера проекта (Project Manager), окно наблюдаемых величин (Watch List) и другие.

Встраиваемое окно можно отличить от обычного по следующим признакам.

- Сокращенная полоса системного меню, включающая обычно только кнопку закрытия окна.
- Наличие в меню, появляющемся при щелчке в окне правой кнопкой мыши, переключателя Dockable — встраиваемое. Если снять метку с этого переключателя, окно перестанет быть встраиваемым. В дальнейшем можно опять установить этот переключатель, и окно снова станет встраиваемым.

При перетаскивании встраиваемого окна размеры его рамки изменяются, если окно перемещается в пределах другого окна.

Встраивание окон позволяет экономить площадь экрана. Для того чтобы переместить встраиваемое окно, надо потянуть курсором мыши за двойную рамку на одной из его границ (на рис. В.21 она расположена на верхней границе окна). При этом можно вынуть его из окна-контейнера и сделать самостоятельным — это так называемое плавающее окно. Для перевода окна в плавающее состояние не обязательно тянуть за двойную рамку, а достаточно произвести на ней двойной щелчок мышью. Можно встроить окно Code Explorer в окно Редактора кода иначе, чем это принято по умолчанию, например, снизу, чтобы не уменьшать видимую длину строк кода. Во встроенном состоянии можно курсором мыши переместить границы окон, практически убрав при желании одно из окон, которое в данный момент не нужно.

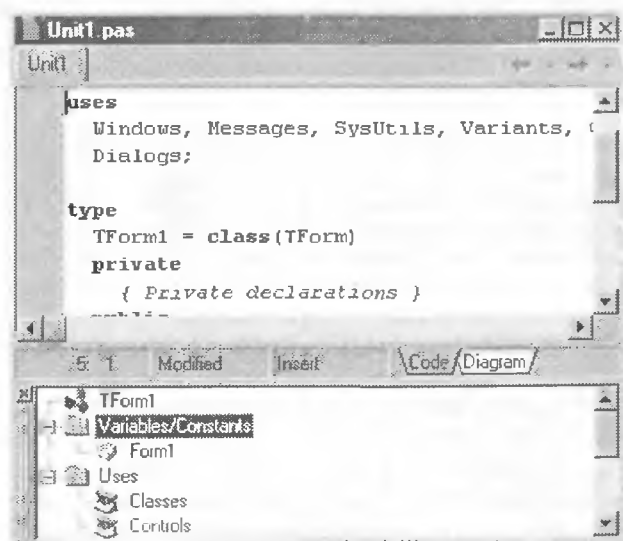


Рис. В.21. Окно Редактора кода с окном Code Explorer, расположенным в нижней части

Очень удобно встраивать окна в Инспектор объектов. Они при этом ложатся на отдельные страницы, совершенно не занимая на экране дополнительного места. А когда требуется, вы всегда можете посмотреть требуемое окно, щелкнув на его закладке. Поскольку не все закладки могут уместиться в заголовке окна, в нем появляются на уровне закладок кнопки со стрелками, направленными влево и вправо. С их помощью можно перейти на страницу, закладка которой не видна.

Технология встраивания окон Drag&Doc реализована в Delphi, начиная с версии 4, в инструментальных панелях. На рис. 14.1 видно, что ИСП содержит 6 панелей, разделенных двойными рамками. Потяните какую-нибудь из панелей (например, палитру компонентов) за эту рамку, и увидите, что ее можно перемещать, например, перевести ее в дополнительный третий ряд панелей, чтобы увеличить доступную длину, или вообще перевести в плавающее состояние.

Среда разработки Delphi 6 поддерживает операции drag-and-drop во время отладки. Например, из редактора кода можно перенести выражение в окно Watch List, после чего это выражение останется в соответствующем списке. Можно перенести выражение в Debug Inspector. Можно также перенести выражение в панель, содержащую дампы памяти в окне CPU, и получить его адрес.

Управление конфигурацией окон

В Delphi имеется много других окон, облегчающих написание кода и отладку приложения. Каждый пользователь открывает те окна, которые требуются ему для того или иного вида работ, создавая удобную для себя конфигурацию окон. Delphi 6 позволяет сохранять сведения о том, какие окна среды разработки открыты, и где именно на экране они расположены. Таких конфигураций может быть создано и сохранено несколько, при этом имеется возможность выделить отдельную конфигурацию, которая будет автоматически загружаться в режиме отладки (так называемый debug desktop) и выгружаться при выходе из него. Для этой цели инструментальная панель содержит кнопки Save current desktop и Set debug desktop, а главное меню — соответствующий подраздел View ► Desktops. Все сохраненные конфигурации среды разработки хранятся в файлах с расширением .dst подкаталога Delphi6\Bin.

Можно сделать так, чтобы при очередном запуске Delphi восстанавливалась конфигурация, которая была установлена на момент завершения предыдущего сеанса работы. Причем не только конфигурация окон, но и загружался проект, с которым вы работали в последний раз. Так что вы сразу можете продолжать работу над тем же проектом. Это делается при помощи команды Tools ► Environment Options.

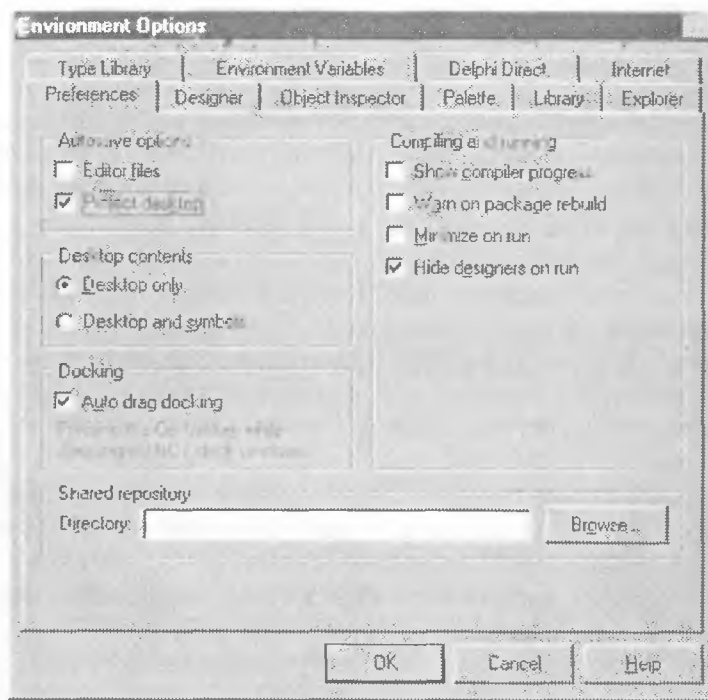


Рис. В.22. Диалоговое окно настроек ИСР

В открывшемся диалоговом окне настроек среды разработки на странице Preferences в группе Autosave options следует включить опцию Project desktop, как показано на рис. В.22. Тогда текущая конфигурация и открытые файлы проекта автоматически сохраняются при завершении сеанса работы с Delphi и автоматически восстанавливаются при начале нового сеанса. При включении этой опции конфигурация окон запоминается также в каждом проекте. Так что если вы впоследствии откроете какой-либо проект, с которым работали ранее, то восстановится конфигурация всех окон, которые были открыты в момент окончания предыдущего сеанса работы с данным проектом.

В конфигурациях сохраняются следующие параметры ИСР:

- совокупность открытых окон;
- размеры и расположение окон на экране (но не сохраняется их положение в так называемой Z-последовательности, определяющей в случае взаимного перекрытия окон, какие из них находятся поверх других);
- состояние каждого окна (свернутое, развернутое, встроенное в другое окно);
- настройки окна (например, в Инспекторе объектов сохраняется способ отображения информации — по алфавиту или по категориям и установки фильтрации, во встраиваемых окнах сохраняется состояние индикатора Docable).

В Delphi 5-6 введены расширенные возможности сохранения конфигураций. Установив на экране некоторую конфигурацию окон, можно выполнить команду View ► Save Desktop. Появится диалоговое окно, показанное на рис. В.23, в котором следует задать имя сохраняемой конфигурации, причем это имя может быть записано русским текстом.

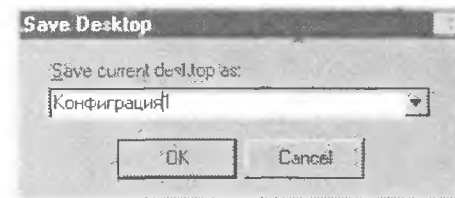


Рис. В.23. Сохранение конфигурации

Эту операцию можно повторять несколько раз для разных конфигураций, которые вы используете. В результате в раскрывающемся списке панели выбора конфигурации появятся составленные вами конфигурации. В дальнейшем, работая с каким-либо проектом, вы сможете в любой момент выбрать в этом списке одну из конфигураций и она появится на экране.

Введенные конфигурации можно впоследствии реорганизовать, удалив ненужные командой View ► Desktops ► Delete. Конфигурация различных вспомогательных окон, установленная для одной из запомненных конфигураций, используется как конфигурация отладки. Какая именно — устанавливается с помощью команды View Desktops ► Set Debug Desktop. Пользователю предлагается список сохраненных конфигураций, из которого он должен выбрать конфигурацию отладки. В этом случае именно эта конфигурация будет автоматически загружаться при запуске приложения на выполнение в режиме отладки. Если, например, вы создали и сохранили две конфигурации: одну, не содержащую вспомогательных окон, сохранили под именем «Default», вторую, с окном наблюдения Watches, встроенным в окно Инспектора объектов, сохранили под именем «Watches». Если, выполняя создание конфигурации отладки приложения, вы указали в качестве конфигурации вариант «Watches», то независимо от того, какую из конфигураций — «Умолчание» или «Watches» вы указали при работе с проектом, в момент запуска приложения на выполнение будет установлена конфигурация «Watches». После окончания сеанса отладки конфигурация автоматически вернется к исходной. Таким образом, интегрированная среда разработки Delphi 6 поддерживает современные технологии эффективной разработки программных продуктов самого разнообразного назначения.

Попов Владимир Борисович

Паскаль и Дельфи. Самоучитель

Главный редактор	<i>Е. Строганова</i>
Заведующий редакцией	<i>И. Корнеев</i>
Руководитель проекта	<i>В. Шачин</i>
Литературный редактор	<i>М. Иванов</i>
Художник	<i>Н. Биржаков</i>
Корректоры	<i>И. Смирнова, Н. Солнцева</i>
Верстка	<i>Р. Гришинов</i>

Лицензия ИД № 05784 от 07.09.01

Подписано к печати 19.01.04. Формат 70×100/16. Усл. п. л. 43,86.

Доп. тираж 4500. Заказ 508.

ООО «Питер Принт», 196105, Санкт-Петербург, ул. Благодатная, д. 67в.

Налоговая льгота — общероссийский классификатор продукции

ОК 005-93, том 2, 95 3005 — литература учебная.

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени «Техническая книга»
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций
190005, Санкт-Петербург, Измайловский пр., 29

Владимир Попов

САМОУЧИТЕЛЬ

Паскаль и Дельфи



Эта книга поможет вам:

- 1 познакомиться с алгоритмическими основами программирования и изучить элементы языка Паскаль;
- 2 освоить принципы структурного программирования, разобраться в объектно-ориентированном программировании;
- 3 научиться работать в среде программирования Дельфи и разрабатывать собственные приложения.

Данная книга представляет собой курс по изучению одного из популярных языков программирования — Паскаль. В нем последовательно излагаются основные принципы структурного программирования, объектно-ориентированного программирования и другие методы разработки приложений на языке Паскаль. Большое внимание уделено изучению интегрированных сред программирования — Турбо Паскаль и Дельфи. Каждая глава книги содержит большое количество подробно разобранных примеров программ. Для самопроверки усвоения теоретического материала вы можете воспользоваться вопросами, имеющимися в конце каждой главы. Выполнение большого количества заданий по разработке приложений в средах программирования поможет сформировать прочные навыки программирования.

Книга предназначена для учащихся школ, студентов высших и средних учебных заведений и благодаря наличию большого количества детально рассмотренных примеров, вопросов и заданий может быть использована для самообразования.

ISBN 5-8046-0156-3



9 785804 601561

Посетите наш web-магазин: www.piter.com

**Изучите один из популярнейших
языков программирования
самостоятельно!**

ПИТЕР
WWW.PITER.COM